



الجامعة الافتراضية السورية
SYRIAN VIRTUAL UNIVERSITY

أسس بناء المترجمات

الدكتور خليل عجمي



Books

أسس بناء المترجمات

الدكتور خليل عجمي

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

هذا الكتاب منشور تحت رخصة المشاع المبدع – النسب للمؤلف – حظر الاشتقاق (CC– BY– ND 4.0)

<https://creativecommons.org/licenses/by-nd/4.0/legalcode.ar>

يحق للمستخدم بموجب هذه الرخصة نسخ هذا الكتاب ومشاركته وإعادة نشره أو توزيعه بأية صيغة وبأية وسيلة للنشر ولأية غاية تجارية أو غير تجارية، وذلك شريطة عدم التعديل على الكتاب وعدم الاشتقاق منه وعلى أن ينسب للمؤلف الأصلي على الشكل الآتي حصراً:

خليل عجمي، أسس بناء المترجمات، من منشورات الجامعة الافتراضية السورية، الجمهورية العربية السورية، 2018

متوفر للتحميل من موسوعة الجامعة <https://pedia.svuonline.org/>

Compilers

Khalil Ajami

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

Published under the license:

Creative Commons Attributions- NoDerivatives 4.0

International (CC-BY-ND 4.0)

<https://creativecommons.org/licenses/by-nd/4.0/legalcode>

Available for download at: <https://pedia.svuonline.org/>



الفهرس

١ الفصل الأول
١ مقدمة
١ ١. ماذا تعني الترجمة
٢ ٢. مراجع
٣ الفصل الثاني
٣ بنية مترجم
٣ ١. مرحلة التحليل
٣ a. التحليل المفرداتي
٣ b. التحليل القواعدي
٤ c. التحليل الدلالي
٤ ٢. مرحلة التركيب والتوليد
٤ a. توليد الرماز
٤ b. أمثلة الرماز
٥ ٣. مراحل موازية:
٥ a. إدارة جدول الرموز
٥ b. إدارة الأخطاء
٦ الفصل الثالث
٦ التحليل المفرداتي
٦ ١. المفردات
٦ a. التعبير المنتظمة
٨ ٢. تنفيذ التحليل المفرداتي
٨ a. توصيف المفردات
٩ b. تحديد الرموز
٩ c. تحليل كلمات النص البرمجي
١١ ٣. أخطاء المفردات
١٢ ٤. تمارين

١٣	الفصل الرابع
١٣	أداة التحليل المفرداتي (f)lex (أنظر الأدوات المرافقة)
١٣	١. بنية ملف توصيف المفردات
١٣	٢. التعابير المنتظمة الخاصة بالأداة (f)lex
١٤	٣. المتحولات والإجرائيات المعرفة مسبقاً في الأداة (f)lex
١٥	٤. خيارات الترجمة
١٥	٥. نماذج عن ملف *.L
١٧	الفصل الخامس
١٧	اللغات الصورية والأوتومات
١٧	١. الأوتومات المنتهي
١٨	٢. تحويل تعبير منتظم إلى أوتومات منته لاحقاً
١٨	٣. تحويل أوتومات منته لاحقاً إلى أوتومات منته حتمياً
٢١	٤. تحويل أوتومات منته إلى تعبير منتظم
٢١	٥. الأوتومات ذات المكس
٢٣	٦. تمارين
٢٥	الفصل السادس
٢٥	التحليل القواعدي
٢٥	١. النحو الصرفي ومفهوم شجرة الاشتقاق
٢٥	a. النحو الصرفي
٢٦	b. شجرة الاشتقاق
٢٩	٢. تنفيذ التحليل القواعدي
٢٩	٣. تحليل نازل من الأعلى إلى الأسفل
٢٩	a. أمثلة
٣١	b. التحليل LL(1)
٣٦	c. النحو LL(1)
٣٧	d. العودية اليسارية (Left Recursion)
٣٩	e. حساب المعاملات اليسارية المشتركة
٤٠	f. النحو التنظيمي
٤٠	g. استنتاج
٤١	٤. تحليل صاعد من الأسفل إلى الأعلى
٤٢	a. خوارزمية التحليل LR:
٤٤	b. بناء جدول التحليل
٤٥	٥. الأخطاء الصرفية (Syntax Errors)
٤٦	a. أسلوب Panic Mode

٤٦b. تصحيح الأخطاء
٤٦c. إضافة قواعد للأخطاء
٤٦ ٦. مسألة
٥٢ الفصل السابع
٥٢ الأداة Yacc/Bison (أنظر الأدوات المرافقة)
٥٢ ١. بنية ملف توصيف القواعد الصرفية
٥٣ ٢. التواصل مع محلل المفردات: yyval
٥٣ ٣. المتحولات، والإجرائيات، والتوابع المعرفة
٥٤ ٤. حالات التضارب Shift/Reduce والتضارب Reduce/Reduce
٥٤ ٥. تحديد أفضليات القواعد وطرق تجميعها
٥٥ ٦. نموذج عن ملف *.Y
٥٦ الفصل الثامن
٥٦ التحليل الدلالي
٥٦ ١. مجال تعريف ورؤية المتحولات
٥٧ ٢. التحقق من الأنماط
٦٠ الفصل التاسع
٦٠ توليد الرماز وأمثله
٦٠ ١. البنية الوسيطة
٦٠ ٢. تنظيم الذاكرة وتنفيذ عملية الحساب
٦٢ ٣. توليد الرماز المقابل للتعليمات
٦٣ ٤. الأمثلة
٦٣ a. تنفيذ مباشر للعمليات على القيم الثابتة اعتباراً من مرحلة الترجمة
٦٣ b. حذف العمليات غير المجدية
٦٤ c. حساب التعابير المشتركة

معجم المصطلحات:

انكليزي	عربي
Compiler	مترجم
Source Program	برنامج مصدري
Destination program	برنامج هدف
Lexical Analysis	التحليل المفرداتي
Tokens	مفردات
Alphabet	أبجدية
Word	كلمة
Concatenation	دمج تسلسلي
Language	لغة
Regular Language	لغة منتظمة
Regular Expression	تعبير منتظم
Lexical Errors	أخطاء المفردات
Syntactical Analysis	التحليل القواعدي
Grammar	نحو صرفي
Derivation Tree	شجرة الاشتقاق
Top-down Parsing	التحليل النازل
Bottom-Up Parsing	التحليل الصاعد
Semantic Analysis	التحليل الدلالي
Code Generation	توليد الرماز
Code Optimization	أمثلة الرماز
Symbol Table	جدول الرموز
Automaton	أوتومات
Formal Languages	لغات صورية
Regular Expression	تعبير منتظم
Finite Automaton	الأوتومات المنتهي
Deterministic Finite Automaton	الأوتومات المنتهي الحتمي
Non-Deterministic Finite Automaton	الأوتومات المنتهي اللاحتمي
Push-Down Automaton	الأوتومات ذات المكس

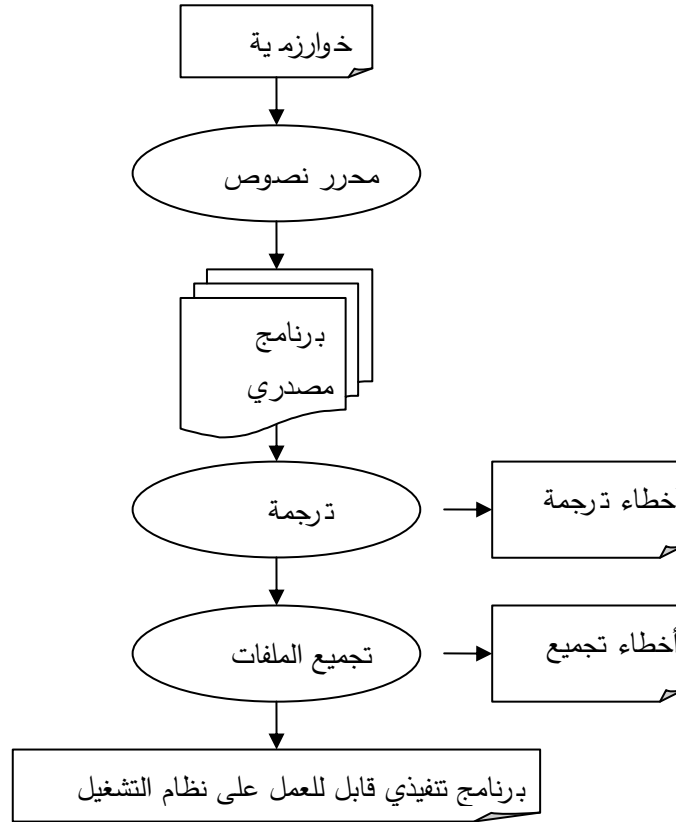
Context-free Language	لغة خارج السياق
Syntax Rules	القواعد الصرفية
Grammar	النحو الصرفي
Derivation	اشتقاق
Derivation Tree	شجرة الاشتقاق
Top-Down Parsing	تحليل نازل
Bottom-Up Parsing	تحليل صاعد
Leftmost Derivation	الاشتقاق اليساري
Rightmost Derivation	الاشتقاق اليميني
Ambiguity	غموض
First	مجموعة الرموز الأولى
Follow	مجموعة الرموز اللاحقة
Terminal Symbol	رمز أولي
Non Terminal Symbol	رمز وسيط
Starting Symbol	رمز البداية
Left Scanning	المسح من اليسار
Right Scanning	المسح من اليمين
Left Recursion	عودية يسارية
Clean Grammar	نحو نظيف
Shift/Reduce	سحب/اختصار
Syntax Error	خطأ صرفي
Symbol Table	جدول الرموز
Code Generation	توليد الرماز
Code Optimization	أمثلة الرماز
Intermediate Structure	البنية الوسيطة
Stack	المكدس
Heap	المكوم
Stack Machine	الآلة ذات المكدس
Register Machine	الآلة ذات السجلات
Virtual Machine	آلة افتراضية

الفصل الأول

مقدمة

1 . ماذا تعني الترجمة

يستخدم أي مبرمج أداة ضرورية جداً في عملية البرمجة، ندعوها المترجم. يمكننا تعريف المترجم بأنه برنامج حاسوبي يترجم النص البرمجي الذي نكتبه بلغة برمجة عالية المستوى (C, Pascal, C++, C#, Java, ...etc) إلى مجموعة تعليمات قابلة للتنفيذ من قبل الحاسوب. تكون هذه التعليمات التنفيذية مكتوبة بلغة منخفضة المستوى سواء كانت لغة ثنائية مؤلفة من أصفار ووحدان (0,1)، أو لغة تجميع. تمر عملية تشغيل برنامج حاسوبي بمجموعة من المراحل التي نمثلها في الشكل التالي:



تجري كتابة النص البرمجي (أو النصوص البرمجية) لأي برنامج حاسوبي، باستخدام محرر نصوص (ضمن ملف واحد أو ضمن مجموعة من الملفات). ندعو هذه النصوص البرمجية التي تؤلف برنامج حاسوبي، بالبرنامج المصدري (Source Program).

يمر البرنامج المصدري بعد ذلك، بمرحلة ترجمة (Compilation) وتجميع (Linking) يجري فيها ربط الملفات الحاوية على البرنامج المصدري الواحد ببعضها البعض وترجمتها إلى مجموعة من التعليمات التنفيذية المكتوبة بلغة منخفضة المستوى.

تشكل التعليمات التنفيذية الناتجة برنامج جديد ندعوه البرنامج الهدف (Destination Program). يأخذ شكل ملف تنفيذي قابل للتشغيل مباشرةً على نظام التشغيل.

تظهر خلال مراحل الترجمة والتجميع أخطاء ندعوها أخطاء الترجمة (تكون ناجمة عن أخطاء في نصوص البرنامج المصدري) أو أخطاء التجميع (تكون ناجمة عن أخطاء في ربط الملفات الحاوية على النصوص). تؤدي هذه الأخطاء إلى توقف عملية الترجمة حتى يجري تصحيحها من قبل المبرمج، قبل إعادة تشغيل المترجم لتوليد "برنامج هدف" خال من الأخطاء.

تجدر الإشارة إلى أن عملية بناء المترجم (والذي عرفناه كبرنامج حاسوبي) تتعلق بعنصرين اثنين بآن واحد:

١- لغة البرمجة المصدرية عالية المستوى الذي يستخدمها المبرمج.

٢- نظام التشغيل الذي سيجري تشغيل البرنامج عليه.

فعلى سبيل المثال، يختلف مترجم لغة C++ الذي يعمل على نظام Windows عن مترجم لغة C++ الذي يعمل على نظام Linux، نظراً لضرورة توليد تعليمات تشغيل تنفيذية مختلفة في هاتين الحالتين، بالرغم من أننا نتكلم عن نفس اللغة وهي C++. في حين، يختلف مترجم لغة C++ عن مترجم لغة Pascal حتى ولو كان المترجمان يعملان على نظام Windows، نظراً لأننا نترجم لغتين برمجيتين مختلفتين.

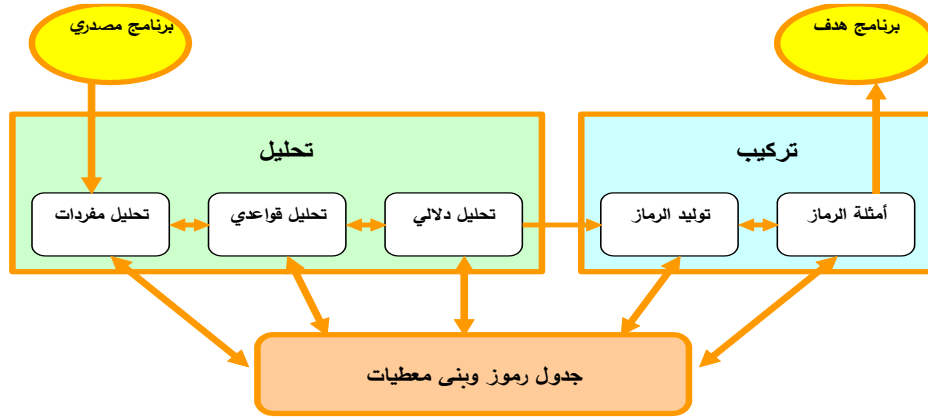
٢. مراجع

- A. Aho, R. Sethi, J. Ulman, **Compiler principles**, InterEdition, 1991
- N. Silverio, **Compiler Construction**, Eyrolles, 1995.

الفصل الثاني

بنية مترجم

تتألف عملية الترجمة من مرحلتين أساسيتين: مرحلة التحليل والتي يجري فيها تقسيم النص البرمجي إلى كلمات وجمل والتأكد من صحتها ودلالاتها، ومرحلة التركيب التي يجري فيها تركيب نص برمجي جديد بنفس دلالة النص المصدري ولكن بلغة أخرى هي اللغة التي تتألف منها التعليمات التنفيذية التي يفهمها نظام التشغيل. يوضح الشكل التالي مكونات المرحلتين:



١. مرحلة التحليل

a. التحليل المفرداتي

تجري في هذه المرحلة عملية التعرف على أنواع الكلمات المؤلفة للنص البرمجي المصدري. لذا تجري قراءة النص البرمجي المصدري حرفاً حرفاً من اليسار إلى اليمين، ويجري تجميع الحروف لتشكيل كلمات. يتولى التحليل المفرداتي المهام التالية:

- i. حذف كافة المحارف التي لا تدخل في صلب النص البرمجي مثل الفراغات، التعليقات، ... الخ.
- ii. تجميع المحارف في كلمات وتحديد نوع كل كلمة: كلمة مفتاحية من لغة البرمجة، متحول، ثابت، قيمة عددية، سلسلة محارف، عملية حسابية، عملية منطقية، ... الخ.

b. التحليل القواعدي

ويمكن أيضاً تسميته بالتحليل الصرفي. إذ يجري خلال التحليل القواعدي جمع الكلمات الناتجة عن التحليل المفرداتي، في جمل وبنى قواعدية تشكل بنية النص البرمجي. يقوم المحلل القواعدي بالتأكد من سلامة الجمل المبنية بالنسبة لقواعد صرفية خاصة بكل لغة برمجة. فعلى سبيل المثال، تحدد القواعد الصرفية طريقة بناء الجملة الشرطية في لغة البرمجة (مثل لغة ++C)، أو طريقة بناء حلقة تكرار فيها.

تشبه عملية التحليل القواعدي للغة برمجة عملية الإعراب في اللغات الطبيعية. إذ يمكن لجملة عربية أن تكون فعلية مؤلفة من فعل وفاعل ومفعول به، أو اسمية مؤلفة من مبتدأ وخبر. كذلك هو الحال في جملة شرطية مكتوبة بلغة برمجة مثل لغة C. إذ يجب أن تبدأ هذه الجملة بكلمة "if" يليها تعبير شرطي يعبر عن شرط معين مثل " $X > 0$ " ومن ثم مجموعة من العمليات التي يجب تنفيذها عند تحقق الشرط، ويمكن أن تتبع هذه العمليات كلمة "else" لتعريف العمليات التي تجري عند انقضاء الشرط، أو أن تنتهي الجملة الشرطية دون كلمة "else".

c. التحليل الدلالي

تجري في هذه المرحلة عملية التحقق من دلالة الجمل المركبة بعد أن تم التأكد من سلامتها قواعدياً. فعلى سبيل المثال، تعتبر عملية التحقق من الأنماط، مرحلة مهمة من مراحل التحليل الدلالي، حيث يجري فيها التأكد من أن عملية مثل ($X * Y$) لها دلالة إذا كانت كل من X و Y تعبر عن قيمتين رقميتين، في حين تصبح هذه العملية دون دلالة (إلا في حالات خاصة ليست في صلب دراستنا هنا) في حال كانت X تعبر عن قيمة رقمية و Y تعبر عن سلسلة محارف.

٢. مرحلة التركيب والتوليد

a. توليد الرماز

تجري فيها عملية توليد التعليمات التنفيذية التي لها نفس دلالة تعليمات النص البرمجي المصدر ولكن بلغة مفهومة من كل من الجهاز ونظام التشغيل اللذين سيجري تشغيل البرنامج عليهما. يمكن في بعض الأحيان توليد الرماز بلغة أخرى عالية المستوى (في حال كان المطلوب هو نقل نصوص برمجية مكتوبة بلغة برمجة قديمة إلى نصوص برمجية مكتوبة بلغة برمجة أحدث لتسهيل الربط مع برامج أخرى)، كما يمكن توليد الرماز بلغة خاصة بآلة افتراضية تعمل على نظام تشغيل كما هو الحال في آلة Java الافتراضية (Java Virtual Machine) التي تعمل على نظام التشغيل ويندوز أو غيره، أو بلغة آلة خاصة بنوع معين من المعالجات وضمن بيئة نظام تشغيل كما هو الحال في بيئة التشغيل Windows التي تعمل على معالجات الحواسيب الشخصية X86.

b. أمثلة الرماز

تهتم هذه المرحلة باختصار وتحسين الرماز المولد وذلك بهدف تسريع عمل البرنامج التنفيذي الناتج وضمان تنفيذ سريع وفعال له عند تشغيله. يجري في هذه المرحلة حذف تعليمات لا معنى لها مثل تعريف متحول وعدم استخدامه، ضرب قيمة ما ب ١، جمع قيمة ما مع ٠ ... الخ.

٣. مراحل موازية:

a. إدارة جدول الرموز

يشكل جدول الرموز أحد أهم بنى المعطيات المستخدمة في المترجمات. إذ يجري تخزين المتحولات المعرفة ضمن النص البرمجي المصدر في جدول الرموز، كما يجري تخزين أنماطها التي جرى الإعلان عنها في النص البرمجي.

يستخدم جدول الرموز أيضاً لتحديد مجال رؤية أو مدى المتحول، وهو مجموعة مقاطع النص البرمجي وإجراءاته التي يمكن فيها استخدام هذا المتحول وفقاً لتعريفه في جدول الرموز. فعلى سبيل المثال، في بعض لغات البرمجة وعند الإعلان عن المتحول X كمتحول محلي من النمط Integer ضمن الإجرائية Proc، لا يمكن للمبرمج استخدام هذا المتحول ضمن النص البرمجي خارج نطاق الإجرائية Proc. لذا يقوم المترجم بالاعتماد على جدول الرموز لتخزين اسم المتحول ونمطه ومداه وذلك بهدف التأكد من عدم استخدام هذا المتحول خارج Proc.

b. إدارة الأخطاء

تنتج عن عمليات التحليل والتركيب السابقة أخطاء ارتكبها المبرمج أثناء كتابته للنص البرمجي: منها أخطاء في كتابة الكلمات، ومنها أخطاء في بناء الجمل، وبعضها أخطاء دلالية ... وهكذا دواليك. يجري التعامل مع كل نوع من أنواع الأخطاء بشكل مختلف ولكن يبقى الهدف الأول والأخير لمعالجة الأخطاء هو إعطاء المبرمج إشارة إلى الخطأ، وتوضيح سبب الخطأ (قدر الإمكان)، ومحاولة تجميع أكبر عدد من الأخطاء الناجمة عن خطأ أول وإظهارها بأن واحد وذلك بهدف تسريع عملية الترجمة.

الفصل الثالث

التحليل المفرداتي

يشكل المحلل المفرداتي الجزء الأول من المترجم وينفذ المرحلة الأولى من عملية الترجمة. تتلخص المهمة الأساسية للمحلل اللفظي بتجميع محارف الدخل، الآتية من النص البرمجي المصدر، بهدف توليد مجموعة من الكلمات التي ستؤلف بدورها جمل يعالجها المحلل القواعدي. ينفذ المحلل المفرداتي أيضاً مجموعة من المهام الثانوية من أهمها: حذف المحارف التي لا دور لها: كالفراغ وسلاسل التعليقات، كما يتولى مهمة حفظ أرقام أسطر النص البرمجي التي يمكن الاستعانة بها للإشارة إلى مكان الأخطاء الموجودة في النص البرمجي. سنستعرض في هذا الفصل بنية المحلل المفرداتي وتفاصيل عملية التحليل المفرداتي. تجدر الإشارة إلى أن مسألة تصميم وتشغيل محلل مفرداتي تكافئ مسألة تصميم برنامج ينفذ عمليات على سلاسل من المحارف ويهتم بالتعرف على أشكال وصيغ محددة لهذه السلاسل، وهي مسألة تصب في مجال بناء المنظومات التي تساعد في البحث عن المعلومات وتشكل صلب محركات البحث.

1. المفردات

في كل نص برمجي تشكل كل مجموعة من الأحرف المتتالية، "كلمة". يكون لكل كلمة من الكلمات المستخدمة في النص البرمجي شكل يحدد انتماءها إلى نوع من الكلمات أو ما ندعوه "نموذج" (Model). ونستخدم لمجموعة الكلمات التي لها نموذج محدد اسم ندعوه "مفردة" (Token). فعلى سبيل المثال: تعبر الكلمات: (+، -، *، /، %) عن عمليات حسابية. يمكن أن نستخدم للتعبير عن العمليات الحسابية المفردة ARTHOP (من Arithmetic Operator). كما تعبر الكلمات (X، Counter، Y، Toto) عن متحولات يستخدمها المبرمج، ويمكن أن نستخدم المفردة IDENT (من Identifier) للتعبير عن المتحولات. كما يكون للمتحولات نموذج يعبر عن شكل المتحول وهو ما سنراه لاحقاً (فعلى سبيل المثال يفرض النموذج الذي يعرّف شكل المتحولات عدم استخدام أرقام لتعريف متحول. إذ لا يمكن للمحلل المفرداتي اعتبار ٥٥ متحول، فالمحلل يعتبر ٥٥ قيمة رقمية).

a. التعبيرات المنتظمة

نستعرض في هذه الفقرة مجموعة من التعاريف والمبادئ الخاصة بنظرية اللغات، وهي تعاريف ومبادئ مُستخدمة بكثافة في عملية التحليل المفرداتي.

تعريف: ندعو "أبجدية" (Alphabet)، مجموعة منتهية غير خالية Σ من الرموز. كما ندعو "كلمة" (Word) كل سلسلة منتهية من عناصر Σ .

نستخدم الرمز ϵ للدلالة على الكلمة الفارغة. كما نستخدم الرمز Σ^* للدلالة على المجموعة غير المنتهية التي تضم جميع الكلمات الممكنة المبنية اعتباراً من الأبجدية Σ . ونستخدم الرمز Σ^+ للدلالة على مجموعة الكلمات غير الفارغة التي يمكن أن نبنيها اعتباراً من Σ . بالنتيجة يكون: $\Sigma^+ = \Sigma^* - \{\epsilon\}$.

يشير الرمز $|m|$ إلى طول الكلمة m ، ويعبر عن عدد الأحرف المنتمية إلى الأبجدية Σ والتي تشكل الكلمة m .
ونستخدم الرمز Σ^n للدلالة على مجموعة الكلمات المنتمية إلى Σ^* والتي طولها n . بالتالي يمكننا أن نستنتج أن:

$$\Sigma^* = \sum_{n=0}^{\infty} \Sigma^n$$

تعريف: نعرف عملية "الدمج التسلسلي" (Concatenation) والتي نرمز لها بالرمز " \bullet "، بأنها عملية نطبقها على كلمتين كالتالي:
لتكن لدينا الكلمة $u = u_1 \dots u_n$ (حيث $u_i \in \Sigma^*$) والكلمة $v = v_1 \dots v_p$ (حيث $v_i \in \Sigma^*$)، يكون حاصل الدمج التسلسلي للكلمتين u و v ، الكلمة $u \bullet v = u_1 \dots u_n v_1 \dots v_p$. بشكل عام يمكن إهمال رمز الدمج التسلسلي " \bullet " وكتابة uv بدلاً عن $u \bullet v$.

فعلى سبيل المثال، لنأخذ الأبجدية $\Sigma = \{a, b, c\}$ ، والكلمات u, v, w, t ، حيث $u = aaba$ ، $v = bbbacbb$ ، $w = c$ ، $t = \epsilon$ هي كلمات تنتمي إلى Σ^* وبأطوال: $\epsilon, 1, 7, 4$ على التسلسل. تكون الكلمة $u \bullet v = aababbacbb$ حاصل الدمج التسلسلي للكلمتين u و v .

خصائص:
$ u \bullet v = u + v $ ✓
$(u \bullet v) \bullet w = u \bullet (v \bullet w)$ ✓
$u \bullet \epsilon = \epsilon \bullet u = u$ ✓

تعريف: ندعو لغة مبنية على أبجدية Σ ، كل مجموعة جزئية من Σ^* .

على سبيل المثال، يمكن اعتباراً من الأبجدية $\Sigma = \{a, b, c\}$ تعريف لغة غير منتهية L_1 حيث تتألف هذه اللغة من الكلمات: $\{\epsilon, a, b, aab, bbcaa, bbccacccaaaabbbcaa, \dots\}$

تعريف: العمليات على اللغات
$L_1 \cup L_2 = \{w : w \in L_1 \text{ OR } w \in L_2\}$ الاجتماع:
$L_1 \cap L_2 = \{w : w \in L_1 \text{ AND } w \in L_2\}$ التقاطع:
$L_1 L_2 = \{w = w_1 w_2 : w_1 \in L_1 \text{ AND } w_2 \in L_2\}$ الدمج التسلسلي:
$L^* = \sum_{n \geq 0} L^n$ الإغلاق:

السؤال الذي يطرح نفسه الآن: في حال كان لدينا لغة ما L . كيف يمكن توصيف الكلمات المنتمية إلى اللغة؟ وكيف يمكن توصيف هذه اللغة. عموماً، توجد عدة أنماط من اللغات وهو ما سنراه في فصل لاحق، ولكن سنركز في هذا الفصل على اللغات التي ندعوها لغات منتظمة.

تعريف: نستطيع تعريف "لغة منتظمة" L (Regular Language) على أبجدية Σ بشكل عودي كما يلي:
 $\{\epsilon\}$ هي لغة منتظمة على Σ .
 إذا كان a حرف من حروف الأبجدية Σ ، تكون $\{a\}$ لغة منتظمة على Σ .
 إذا كانت L لغة منتظمة على Σ ، تكون كل من L^* و L^n لغات منتظمة على Σ .
 إذا كان كل من L_1 و L_2 لغات منتظمة، تكون كل من $L_1 \cup L_2$ و $L_1 L_2$ لغات منتظمة على Σ .
 لا توجد لغات منتظمة أخرى على Σ .

تعريف: يمكن توصيف اللغات المنتظمة، باستخدام أداة ندعوها "التعابير المنتظمة" (Regular Expressions).
 نعطي فيما يلي تعريف عودي للتعبير المنتظمة:
 ϵ هو تعبير منتظم يوصف اللغة $\{\epsilon\}$.
 \checkmark إذا كان a حرف من حروف الأبجدية Σ ، يكون a تعبير منتظم يوصف اللغة $\{a\}$.
 \checkmark إذا كان r تعبير منتظم يوصف اللغة L فإن كل من $(r)^*$ و $(r)^+$ عبارة عن تعبيرين منتظمين يوصفان اللغتين L^* و L^+ على الترتيب.
 \checkmark إذا كان r_1 و r_2 تعبيرين منتظمين يوصفان اللغتين L_1 و L_2 على الترتيب، فإن كل من $(r_1)(r_2)$ و $(r_1) | (r_2)$ عبارة عن تعبيرين منتظمين يوصفان اللغتين $L_1 L_2$ و $L_1 \cup L_2$ على الترتيب.
 \checkmark لا توجد تعابير منتظمة أخرى.

أمثلة عن التعابير المنتظمة:

- \checkmark يوصف $(a | b)^*$ مجموعة الكلمات (اللغة) المولفة من a و b ، أو الكلمة الفارغة.
- \checkmark يكون $(b | a)^* = (a | b)^*$
- \checkmark يوصف $(a | b)^* bbb(a | b)^*$ مجموعة الكلمات (اللغة) التي تحتوي على السلسلة bbb فيها.

٢ . تنفيذ التحليل المفرداتي

a. توصيف المفردات

توفر لنا التعابير المنتظمة أداة لوصف الكلمات التي لها نموذج محدد. يتم التعبير عن هذا النموذج بمفردة. فعلى سبيل المثال، جميعنا يعلم أن استخدام اسم أي متحول في نص برمجي يخضع لقواعد محددة:

- \checkmark أن يبدأ الاسم بحرف ومن ثم نستخدم أي حرف، أو رقم، بالإضافة إلى المحرف " _ " (underscore)، ودون أن يكون هناك حدود لطول اسم المتحول يمكن استخدام الأحرف الصغيرة (small letters) أو الكبيرة (capital letters).

وعليه يمكننا أن نعرّف المفردة IDENT المعبرة عن المتحولات، وذلك اعتماداً على تعبير منتظم، كما يلي:
 IDENT="(a|b|c ...|y|z|A|B|C....|Y|Z)(a|b|c ...|y|z|A|B|C....|Y|Z|0|1|2|3|4|5|6|7|8|9)*"

لاختصار العمليات يمكننا أن نعرف ما يلي:

Letter = a – z | A – Z

Digit = 0 – 9

Sep = _

EndOfString="*"

وبالتالي يصبح تعريف IDENT على الشكل التالي:

IDENT = Letter (Letter | Digit | Sep)* EndOfString

تجدر الإشارة إلى عدم إمكانية تمثيل كل نماذج الكلمات (وبالتالي عدم إمكانية تعريف المفردات المعبرة عنها) باستخدام التعابير المنتظمة. فعلى سبيل المثال، في حال كان لدينا في لغة معرفة على أبجدية {a,b} نموذج كلمات له الشكل $a^n b^n$ يتبع القاعدة التالية: "تبدأ الكلمة بالحرف a الذي يتكرر n مرة ومن ثم يتبعها نفس العدد من b الذي سيظهر n مرة أيضاً من أجل أي n). فإننا ببساطة لا نستطيع استخدام التعابير المنتظمة للتعبير عن هذا النموذج (سيتم توضيح السبب في فصل لاحق).

b. تحديد الرموز

تنقسم رموز لغة برمجة إلى:

✓ الكلمات المفتاحية الأساسية مثل if، while، ... بالإضافة إلى الرموز البسيطة كالعلاقات الحسابية (+، -، *، /، %) أو علاقات المقارنة (>، <، >=، <=) وغيرها، فإننا نكتفي بتعريف المفردة المعبرة عنها مثل ARTHOP التي تعبر عن العلاقات الحسابية.

✓ بالنسبة للرموز التي تحتاج لقواعد محددة لكتابتها (لها نموذج) مثل المتحولات، فإننا (وكما أوضحنا في الفقرة السابقة) نحدد قواعد تعريف نموذجها ونعطي تعريف للمفردات المعبرة عنها كما هو الحال مع IDENT. بنفس الطريقة يمكن أن نستخدم المفردة REAL للتعبير عن عدد حقيقي (مثل 5.3) والذي تكون قاعدة كتابته كما يلي:

POINT = “.”

REAL = (Digit)* POINT (Digit)*

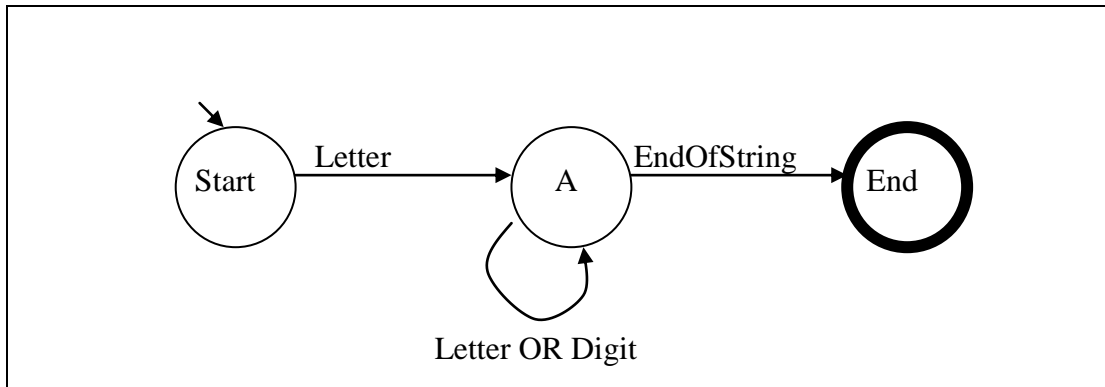
c. تحليل كلمات النص البرمجي

لنفترض أننا نحاول كتابة برنامج للتعرف على أسماء المتحولات بحيث يأخذ البرنامج في دخله اسم ويعيد قيمة 1 أو 0 للدلالة على أن الكلمة تعبر عن اسم متحول أو لا تعبر على الترتيب. لنفترض أن اسم المتحول يتبع للنموذج الذي تمثله المفردة IDENT التي حددناها سابقاً.

نظرية: لكل لغة منتظمة أوتومات منته.

نتيجة: كل أوتومات منته يكافئ برنامج حاسوبي.

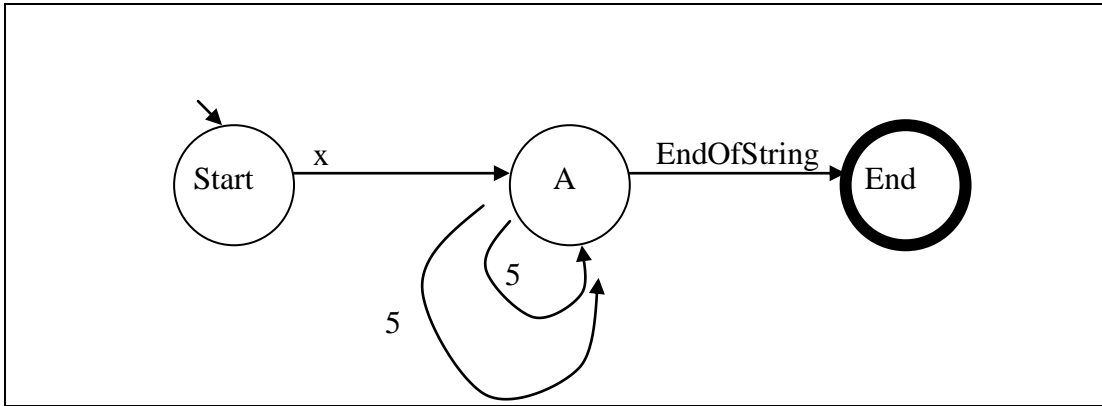
فعلى سبيل المثال، يمكن تمثيل التعبير المنتظم الذي يعرف المفردة IDENT على الشكل التالي:



تعريف: يمكننا تعريف أوتومات منته بشكل أولي وسنعود لهذا الموضوع في فصل لاحق) على أنها بيان من مجموعة منتهية من الحالات والوصلات.

- ✓ هناك أوتومات تتميز بوضع لاصقات على الحالات لتعريفها، وهناك أوتومات تتميز بوضع اللاصقات على الناقلات. في كلا الحالتين نحصل على نفس النتيجة.
- ✓ لكل أوتومات حالة ابتدائية.
- ✓ لكل أوتومات مجموعة من الحالات النهائية التي يمثل الوصول إليها انتهاء عمل الأوتومات
- ✓ يتم تعريف اللاصقات بالاعتماد على مجموعة من الرموز ومجموعة من العمليات.
- ✓ يبدأ عمل الأوتومات اعتباراً من الحالة الابتدائية وتتبع، من أجل كل رمز مقروء على مدخلها، وصلة مرتبطة بها وذات لاصقة تتكون من الرمز المقروء (في حال كانت اللاصقات مثبتة على الناقل).
- ✓ في حال تمثيل الأوتومات لتعبير نظامي نقول عن كلمة أنها مقبولة من أوتومات إذا استطعنا الوصول (انطلاقاً من الحالة الابتدائية) إلى حالة نهائية باستخدام رموز وحروف الكلمة.

فعلى سبيل المثال يتم التأكد من أن الكلمة x55 مقبولة من الأوتومات السابقة الموضحة في الشكل بتمريرها حرفاً حرفاً ابتداءً من الحالة الابتدائية. نلاحظ أنها تصل بعد ثلاث انتقالات إلى الحالة النهائية:



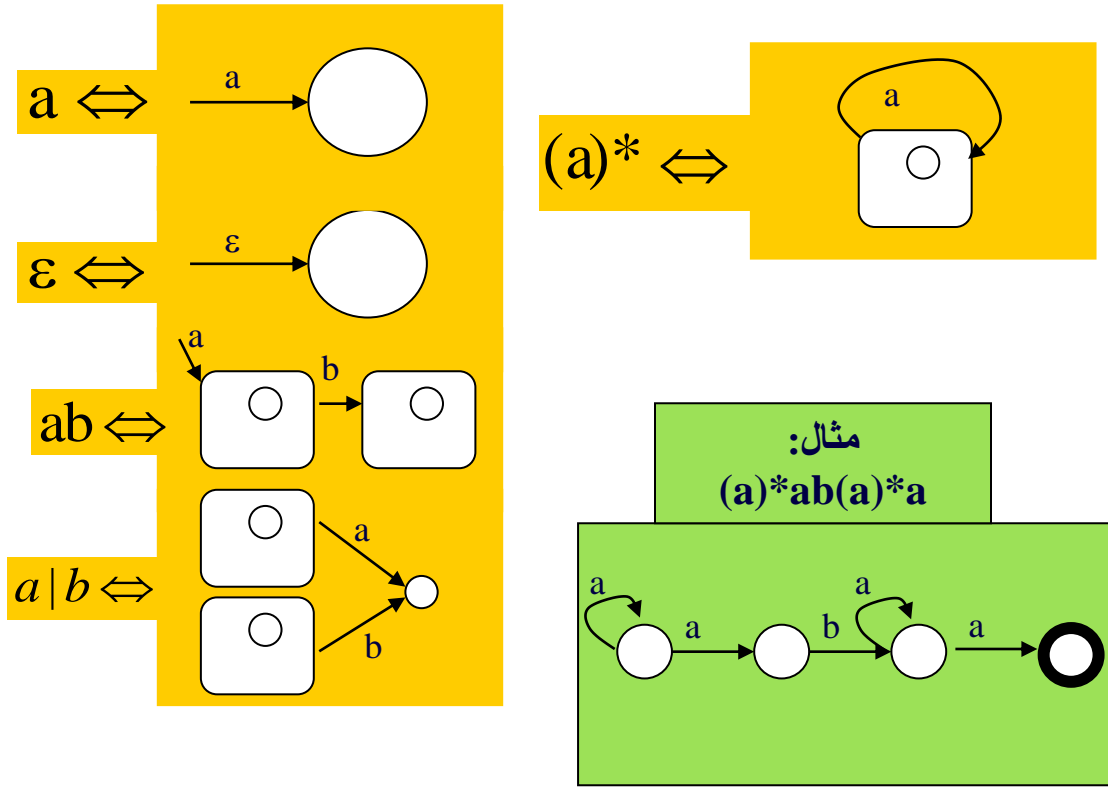
وبما أن كل أوتومات تكافئ برنامج حاسوبي، فيمكننا الآن افتراض وجود إجرائية getNext() تقرأ الحرف التالي من سلسلة الحروف وتضع هذا الحرف في متحول Symbol. ولنكتب إجرائية IsIdentifier() التي تقوم بالتحقق من كون الكلمة المقروءة هي عبارة عن اسم متحول:

```

int IsIdentifier( )
{
    getNext( );
    if (Symbol is a Letter)
    {
        getNext( );
        while ((Symbol is a Letter) || (Symbol is a Digit)) /*Case Switch*/
            getNext( );
        if (Symbol == ' ') /* Space*/
            return TRUE;
        else
            return Error( );
    }
    else
        return Error( );
}

```

لنلق نظرة الآن على قواعد تحويل تعبير منتظم إلى أوتومات والتي يمكن التعبير عنها بخوارزمية تحويل والتي نعرضها هنا بشكل بياني فقط:



- بالنتيجة، نستطيع تحويل تعبير منتظم إلى أوتومات منته حتمي، مما يعني أنه سيكون مكافئ لبرنامج حاسوبي وأننا سنستطيع توليد برنامج حاسوبي. وعليه بإمكاننا أتمتة عملية كتابة المحلل من خلال الخطوات التالية:
- ✓ تعريف التعبيرات المنتظمة المعبرة عن مفردات اللغة التي تمتلك نماذج.
 - ✓ استخدام خوارزمية تحويل تعبير منتظم إلى أوتومات.
 - ✓ توليد البرنامج الحاسوبي المكافئ للأوتومات وهو البرنامج الذي سيقوم بمعالجة النص البرمجي المصدر وقراءته وتحديد أنواع كلماته.

٣. أخطاء المفردات

تكون علاقة المحلل المفرداتي مع الأخطاء قاصرة. إذ يمكن للمحلل المفرداتي التعرف على عدد محدود من الأخطاء:

- ✓ خطأ في كتابة رمز من الرموز التي لا تمتلك نموذج ولا توجد قاعدة لتعريفها، كالرموز البسيطة أو الكلمات المفتاحية.
- ✓ خطأ في تطبيق قاعدة من قواعد تعريف المفردات، ككتابة متحول له الشكل 0.55X، حيث يظهر وضوحاً أن هذا الرمز ليس كلمة مفتاحية، ولا رمز بسيط، ولا يخضع لقاعدة تعريف IDENT (فهو ليس متحول)، ولا لقاعدة تعريف REAL (فهو ليس عدد حقيقي).

تتم معالجة الأخطاء عموماً اعتماداً إما على طريقة تسجيل الحرف الذي سبب الخطأ وإهماله والمتابعة كما سنرى لاحقاً عند التطرق لمعالجة الأخطاء في المحلل القواعدي، حيث سننظر للأمر على نحو أكثر عمومية. أو بمحاولة تصحيح بعض الأخطاء في الرموز البسيطة والكلمات المفتاحية وهو أمر تحاول بعض محررات النصوص المرتبطة بالترجمات تقديمه، كأن يتم تصحيح while بكتابة while في حال أتت هذه الكلمة في بداية الجملة وتلتها جملة تعبر عن حلقة تكرارية. إذ يعتبر محرر النصوص في هذه الحالة أن الكلمة الصحيحة هي الكلمة المفتاحية التي تحتاج لأقل عدد ممكن من التغييرات حتى تتحول من كلمة خطأ إلى كلمة صحيحة في المكان الذي تظهر فيه.

٤. تمارين

السؤال ١:

ماهي اللغات التي توصفها التعابير المنتظمة التالية المُعرّفة على الأبجدية $\{a,b\}$:

$$a(a|b)^*b \quad \checkmark$$

$$aab(aa|bb)^+ \quad \checkmark$$

$$(aa)^*a \quad \checkmark$$

الجواب ١:

- ✓ اللغة التي تبدأ جميع كلماتها بالحرف a وتنتهي بالحرف b.
- ✓ اللغة التي تبدأ جميع كلماتها بالسلسلة aab ومن ثم يليها تكرار واحد على الأقل لثنائية aa أو من bb
- ✓ اللغة التي لا تحتوي كلماتها إلا على حرف a ويكون طول كلماتها مفرداً.

السؤال ٢:

ما هي التعابير المنتظمة التي يمكن أن تعرف اللغات التالية:

$$\checkmark \text{ اللغة على الأبجدية } \{a,b,c\} \text{ والتي تبدأ جميع كلماتها بالحرف } a.$$

$$\checkmark \text{ الأعداد الصحيحة من مضاعفات } 5.$$

الجواب ٢:

$$\checkmark a(a|b|c)^*$$

$$\checkmark (0|1|2|3|4|5|6|7|8|9)^*(5|0)$$

الفصل الرابع

أداة التحليل المفرداتي flex (f) (أنظر الأدوات المرافقة)

تم بناء العديد من الأدوات المساعدة في توليد برنامج التحليل المفرداتي. تقرأ هذه الأدوات توصيف كامل لمختلف مكونات اللغة (من كلمات مفتاحية ونماذج مُعرّفة باستخدام تعابير منتظمة) اعتباراً من ملف دخل، وتولد نص برمجي (بلغة C أو غيرها) قادر على مسح نص برمجي مصدري والتحقق من عدم وجود أي أخطاء في وفقاً للتوصيف المُعطى.

تم بناء أداة lex الخاصة بأنظمة Unix، وتم تطوير أداة مكافئة لها من قبل GNU وتدعى flex (f)، تأخذ ملفات دخل لها شكل محدد وتعطي برنامج بلغة C يمثل المحلل المفرداتي ويمكن ربطه ببقية أجزاء المترجم لبناء مترجم متكامل. يمكن تشغيل الأداة بالشكل:

```
> (f)lex specification_file.l
```

سيتولد نتيجة هذا التشغيل ملف lex.yy.c والذي يجب تمريره على مترجم للغة C مع المكتبة البرمجية الخاصة بالأداة flex (f):

```
> gcc lex.yy.c -l (f)l
```

يحتوي ملف التوصيف (specification_file.l) على تعابير منتظمة تعرف المفردات الممثلة لكل نموذج من نماذج الكلمات (متحول، عدد حقيقي، ... الخ). يعمل المحلل، كما سبق وشرحنا، على تتبع كلمات النص البرمجي حرفاً، حرفاً حتى الوصول إلى أطول سلسلة من الأحرف متطابقة مع أحد التعابير المنتظمة أو أحد الرموز المعروفة.

١ . بنية ملف توصيف المفردات

```
% {  
declaration (in C Language) of variables, constantes, ...  
% }  
declaration of regular definitions  
%%  
Translation rules  
%%  
principal bloc & auxiliary functions (Optional)
```

تجدر الإشارة إلى أن القسم الأخير المتمثل في (principal bloc & auxiliary functions) هو قسم اختياري تتم إضافته في حال كان البرنامج الرئيسي (main) الخاص بالمحلل موجود في هذا الملف.

٢ . التعابير المنتظمة الخاصة بالأداة flex (f)

يتألف تعبير منتظم flex (f) من محارف عادية ومحارف ذات دلالة خاصة وهي:

```
$ ، \ ، ^ ، [ ، ] ، { ، } ، < ، > ، - ، + ، ، * ، / ، | ، ؟
```

فيما يلي التعابير المنتظمة التي يقبلها flex (f) مع الانتباه إلى أن الأداة flex (f) حساسة للفرق بين الأحرف الصغيرة (small letters) والكبيرة (capital letters).

التعبير	ما تفهمه الأداة	مثال
C	حرف غير خاص	&
\c	في حال كان c محرف لا ينتمي إلى مجموعة الأحرف الصغيرة (small letters) من a إلى z، تأخذ الأداة الحرف c كما هو ودون أي اعتبارات خاصة وتعتبره حرف عادي غير خاص.	\+
“s”	سلسلة المحارف s كما هي	“abc*+”
r1r2	r1 متبوعة بـ r2	ab
^	في حال أتى كمحرف أول يسبق تعبيراً آخر، فإنه يُفهم على أن ما سيرد بعده يجب أن يكون في أول السطر	^ abc
\$	في حال أتى كمحرف أخير يلي تعبيراً آخر، فإنه يُفهم على أن التعبير الذي سيرد بعده سيكون في آخر السطر.	abc\$
[s]	أي حرف من الأحرف المؤلفة للسلسلة s	[abc]
[^s]	أي حرف عدا الأحرف المؤلفة للسلسلة s	[^abc]
r*	يظهر التعبير r عدد من المرات تتراوح من 0 إلى اللانهاية	a*
r+	ظهور لمرة واحدة على الأقل للتعبير r	a+
r?	ظهور لمرة واحدة على الأكثر للتعبير r	a?
r{m}	ظهور m مرة للتعبير r	a{3}
r{m,n}	عدد مرات ظهور لـ r يتراوح بين m و n	a{2,5}
r1 r2	r1 أو r2	a b
r1 / r2	r1 في حال كانت متبوعة بـ r2	ab / cd
(r)	r	(a b)? c
\n	سطر جديد	
\t	Tabulation	
EOF	نهاية الملف وهي خاصة بـ (f)lex	

ملاحظات:

- b. لا يمكن للمحارف الخاصة \$، ^، / الظهور ضمن قوسين () ولا ضمن تعاريف منتظمة.
- c. ضمن قوسين من الشكل []، يبقى المحرف \ فقط محرف خاص.
- d. لا يبقى المحرف - محرفاً خاصاً إلا إذا كان في المنتصف أي ليس في البداية ولا في النهاية.

٣. المتحولات والإجرائيات المُعرفة مسبقاً في الأداة (f)lex

العنصر	الاستخدام
char yytext[]	جدول من المحارف يحتوي سلسلة الدخل التي تم قبولها
int yyleng	طول سلسلة الدخل التي تم قبولها
int yylex()	الإجرائية التي يجري عبرها إقلاع المحلل وتستدعي () yywrap
int yywrap()	إجرائية يجري استدعاءها في نهاية سلسلة من محارف الدخل. تعتبر إجرائية مساعدة تعيد 0 في حال كان من الضروري متابعة التحليل (وجود ملف دخل آخر أو سلسلة دخل أخرى)، وتعيد 1 في الحالة المعاكسة.
int main()	يحتوي فقط استدعاء للإجرائية () yylex.

<code>unput(char c)</code>	يحذف حرف من سلسلة الدخل بالعودة حرف إلى الورا وأعتبار الحرف غير مقروء (موجود فقط في lex (f) ولا يوجد في lex).
<code>int ylineno</code>	رقم السطر الحالي الذي تجري معالجته
<code>yyterminate()</code>	إيقاف التحليل (موجود فقط في lex (f) ولا يوجد في lex).

٤ . خيارات الترجمة

الخيار	المعنى
-d	حالة تنقيح (debug mode)
-i	عدم التفريق بين الأحرف الصغيرة (small letters) والأحرف الكبيرة (capital letters)
-l	للتوافق التام مع الأداة lex الخاصة بأنظمة unix
-s	للخروج مباشرة عند مصادفة أي محرف لا يتوافق مع أي رمز أو نموذج

٥ . نماذج عن ملف *.L

يقوم المحلل الموصف في الملف التالي بحساب عدد الأحرف الصوتية وعدد الأحرف غير الصوتية وعدد علامات التنقيط لسلسلة من المحارف المدخلة من لوحة المفاتيح.

```
% {
  int nbvowels, nbconsonant, nbPunct;
% }
consonant    [b-df-hj-np-xz]
punctuation  [,:;?!\\.]
%%
[aeiouy]     nbvowels++;
{consonant}   nbconsonant++;
{punctuation} nbPunct++;
.\\n         // Nothing to do
%%
main(){
  nbvowels = nbconsonant = nbPunct = 0;
  yylex();
  printf("No %d vowels, %d consonants and %d punctuations.\\n",
        nbvowels, nbconsonant, nbPunct);
}
```

يقوم المحلل الموصف في الملف التالي بتعديل الدخل عبر حذف الأسطر التي تبدأ بالحرف p بالإضافة إلى حذف الأعداد الصحيحة، ومن ثم تحويل الحرف 0 إلى * والإبقاء على ما تبقى دون تغيير.

digit	[0-9]
integer	{ digit }+
%%	
{ integer }	printf("I deny the integers");
^p(.)*\n	printf("I hate lines starting with p\n");
.	if (yytext[0]=='o') printf("*") else printf("%c",yytext[0]);

الفصل الخامس

اللغات الصورية والأوتومات

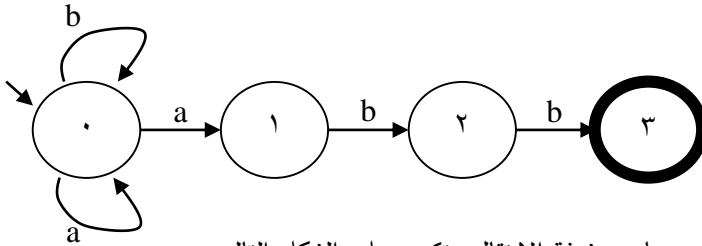
١. الأوتومات المنتهي:

تعريف: نُعرّف "الأوتومات المنتهي" (Finite Automaton)، بأنها خماسية $A = (Q, q_0, F, \Sigma, \Delta)$ حيث:

- Q : مجموعة منتهية من الحالات.
- q_0 : حالة من Q ندعوها الحالة الابتدائية.
- F : مجموعة حالات محتواة في Q وندعوها الحالات النهائية.
- Σ : أبجدية تتألف من مجموعة من الرموز (بما فيها ϵ الرمز الفارغ) والتي ندعوها رموز الدخل.
- Δ : تابع انتقال معرف بالشكل التالي: $\Delta: Q \times \Sigma \rightarrow 2^Q: \Delta(q_i, a) = \{q_{i1}, \dots, q_{in}\}$

مثال: لتكن الأوتومات $A = (\{0,1,2,3\}, 0, \{4\}, \{a, b\}, \Delta)$

$$\left. \begin{array}{l} \Delta(0, a) = \{0,1\} \\ \Delta(0, b) = 0 \\ \Delta(1, b) = 2 \\ \Delta(2, b) = 3 \end{array} \right\} \text{حيث يكون التابع:}$$



وتكون الأوتومات ممثلة بالشكل التالي:

كما يمكن تمثيل التابع السابق بمصفوفة ندعوها مصفوفة الانتقال وتكون على الشكل التالي:

State/ Alphabet	a	b
0	0,1	0
1	—	2
2	—	3
3	—	—

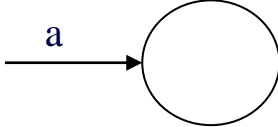
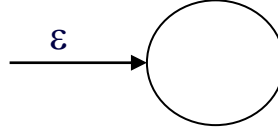
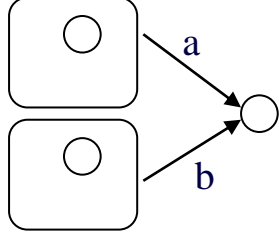
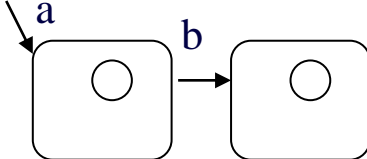
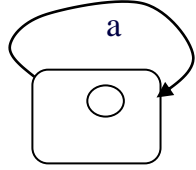
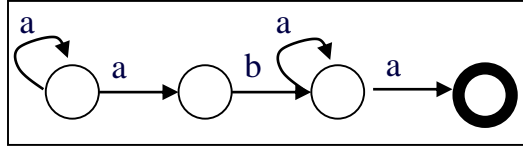
تعريف: نُعرّف "اللغة التي تقبلها أوتومات" بأنها مجموعة السلاسل التي تسمح بالمرور من الحالة الابتدائية إلى إحدى الحالات النهائية للأوتومات.

مثال: في حالة الأوتومات السابقة، تقبل هذه الأوتومات اللغة الممثلة بالتعبير المنتظم: $(a | b)^* abb$

بشكل عام، هناك تقابل بين الأوتومات المنتهي والخوارزمية، فكل أوتومات منته يعبر عن خوارزمية أو عن برنامج منته. وتزداد سهولة التعبير عن أوتومات بخوارزمية، عندما تكون الأوتومات من النوع المنتهي والحتمي وهو ما سنعرّفه لاحقاً.

٢ . تحويل تعبير منتظم إلى أوتومات منته لاحتمي

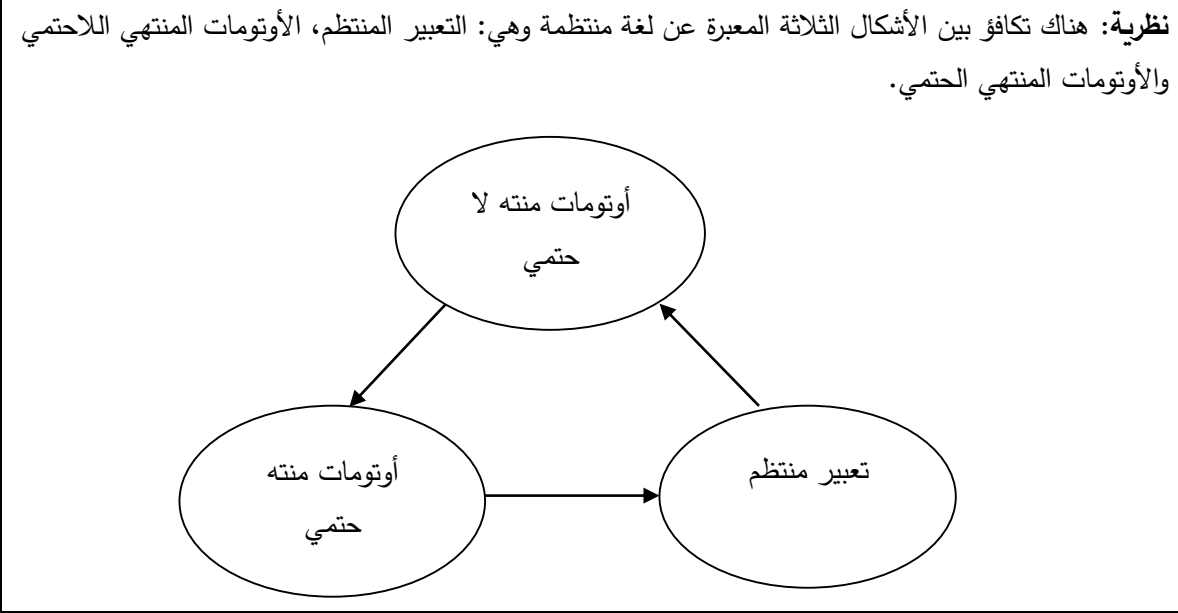
لبناء أوتومات منتهي اعتباراً من تعبير منتظم يمكننا اعتماد الخوارزمية الموضحة في الشكل التالي والتي تحدد الأوتومات المقابل لكل نوع من أنواع التعابير المنتظمة:

$a \Leftrightarrow$  <p>حالة رمز a</p>	$\epsilon \Leftrightarrow$  <p>حالة الرمز الفارغ</p>
$a b \Leftrightarrow$  <p>حالة اختيار بين رمزين أو تعبيرين منتظمين</p>	$ab \Leftrightarrow$  <p>حالة تسلسل رمزين أو تعبيرين منتظمين</p>
$(a)^* \Leftrightarrow$  <p>حالة إغلاق</p>	<p>مثال:</p> <p>$(a)^* a b (a)^* a$</p> 

٣ . تحويل أوتومات منته لاحتمي إلى أوتومات منته حتمي

تعريف: نقول عن "الأوتومات المنتهي" أنه "حتمي" (Deterministic) في حال لم يكن في أبجديته الرمز ϵ وكان تابع الانتقال معرفاً بالشكل: $\Delta: Q \times \Sigma \rightarrow Q: \Delta(q_i, a) = q_j$ بحيث يتم الانتقال بأي رمز من حالة إلى حالة واحدة فقط.

يسمح الانتقال من أوتومات منته لاحتمي إلى أوتومات منته حتمي بتجنب حالات الرجوع إلى الخلف عند التأكد من قبول الأوتومات لكلمة من اللغة المعرفة بهذه الأوتومات. يعود السبب في ذلك إلى عدم وجود عدة خيارات للانتقال من حالة إلى حالة تليها باستخدام نفس الرمز.



يجري تحويل أوتومات منته لاحتمي إلى أوتومات منته حتمي اعتماداً على الخوارزمية التالية:

Input: Non Deterministic Finite Automaton $A1 = (Q1, q_0, F1, \Sigma, \Delta1)$

Output: Deterministic Finite Automaton $A2 = (Q2, q_0, F2, \Sigma, \Delta2)$

For each $p \in Q1$ **and for each** $a \in \Sigma$ **Do**

Add to transition table all the composite states produced by $\Delta1(p, a)$

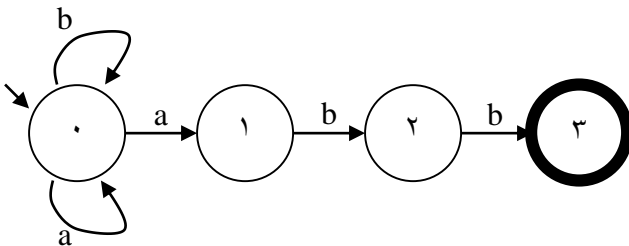
All states with at least one final state, become final states

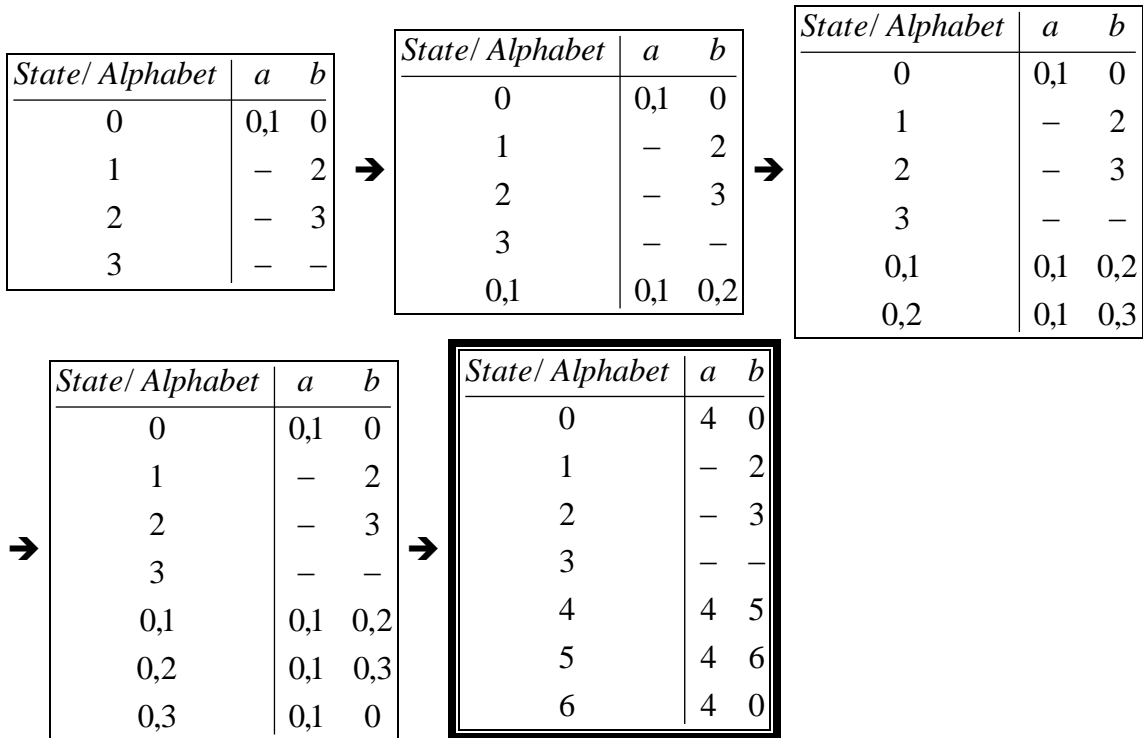
Renumber the states

مثال: نأخذ حالة $A = (\{0,1,2,3\}, 0, \{4\}, \{a,b\}, \Delta)$ حيث التابع:

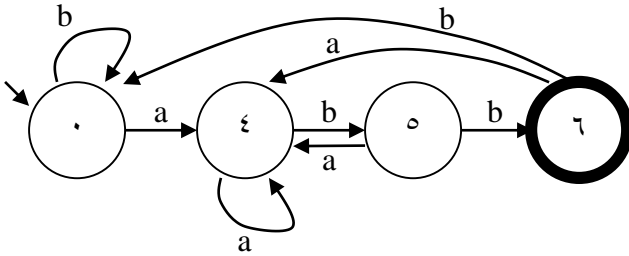
$$\left. \begin{array}{l} \Delta(0, a) = \{0,1\} \\ \Delta(0, b) = 0 \\ \Delta(1, b) = 2 \\ \Delta(2, b) = 3 \end{array} \right\} \text{حيث يكون}$$

للأوتومات الشكل الموضح فيما يلي:

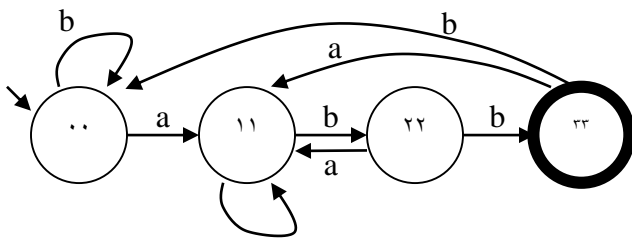




ويكون الأوتومات المنتهي الحتمي الناتج كما يلي:



أما جزء الأوتومات المؤلف من الحالات ١ و ٢ و ٣ فيمكن إهماله لأننا لا يمكن أن نصل إليه اعتباراً من الحالة الابتدائية. لذا يصبح هذا الجزء إضافي ولا معنى له ويمكن إهماله، وتكون الأوتومات الناتجة و المرسومة في الشكل السابق هي الأتومات المنتهي الحتمي الناتج والتي يمكن إعادة ترقيم حالاتها كما يلي:



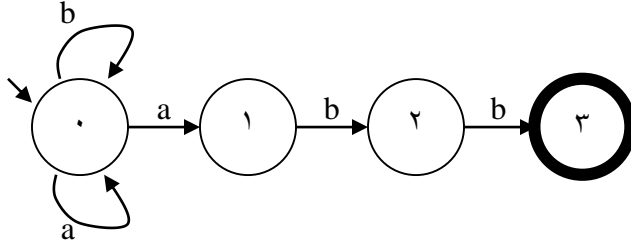
٤ . تحويل أوتومات منته إلى تعبير منتظم

لنفرض أن لدينا لغة L تقبلها الأوتومات $A = (Q, q_0, F, \Sigma, \Delta)$ ولنفرض أن L_i هي اللغة التي يمكن أن تقبلها الأوتومات A إذا اعتبرنا أن الحالة الابتدائية لجميع سلاسلها هي q_i . يمكننا عندها كتابة جملة معادلات التي تربط جميع الأجزاء L_i التي تكون اللغة الأصلية L التي تقبلها A ، كما يلي:

- كل انتقال من الشكل $\Delta(q_i, a) = q_j$ يسمح بكتابة المعادلة: $L_i = aL_j$.
- من أجل كل $q_i \in F$ لدينا المعادلة $L_i = \varepsilon$.
- يمكن تجميع جميع المعادلات $L_i = \alpha$ و $L_i = \beta$ بالشكل $L_i = \alpha | \beta$.
- يتم حل جملة المعادلات الناتجة عن طريق التعويض لحساب L_0 .

خاصة مهمة: $[L = uL | v \Rightarrow L = u^*v]$

مثال: اعتباراً من الأوتومات



يمكننا أن نستنتج جملة المعادلات التالية:

$$\begin{cases} L_0 = aL_0 | bL_0 | aL_1 \\ L_1 = bL_2 \\ L_2 = bL_3 \end{cases} \rightarrow \begin{cases} L_0 = aL_0 | bL_0 | aL_1 \\ L_1 = bbL_3 \end{cases} \rightarrow \{L_0 = aL_0 | bL_0 | abbL_3 \rightarrow (a|b)^*abb$$

٥ . الأوتومات ذات المكس

هناك بعض اللغات التي لا يمكن التعرف عليها باستخدام أوتومات منته ولا يمكن توصيفها باستخدام تعبير منتظم. تكون هذه اللغات أوسع من اللغات المنتظمة وتدعوها باللغات خارج السياق. من الأمثلة النمطية على لغة من هذا النوع، اللغة المبنية على الرمزين a و b ولها الشكل $a^n b^n$ من أجل n صحيح موجب والتي تنتمي إليها سلاسل مثل ab أو $aaabbb$... الخ.

يبدو واضحاً من شكل اللغة $a^n b^n$ أن محاولة تمثيلها بأوتومات ستصطدم بعائق الحاجة إلى وجود ذاكرة تسمح بتخزين عدد المرات (وهي n) التي ظهر فيها الرمز a في الكلمة للتأكد من أن b ستظهر بنفس العدد من المرات، وهو أمر غير ممكن في أي أوتومات منته كون n غير ثابتة وغير محددة ويمكن أن تسعى إلى اللانهاية. لتمثيل مثل هذه اللغات، نستخدم أداة ذات إمكانيات إضافية وهي الأوتومات ذات المكس.

تعريف: نعرّف "الأوتومات ذات المكس" (Push Down Automaton)، عبارة عن سباعية $P = (Q, q_0, F, \Sigma, \Gamma, \tau, \Delta)$ حيث:

- Q : مجموعة منتهية من الحالات.
- q_0 : حالة من Q ندعوها الحالة الابتدائية.
- F : مجموعة حالات محتواة في Q وندعوها الحالات النهائية.
- Σ : الأبجدية وتتألف من مجموعة من الرموز.
- Γ : أبجدية المكس.
- τ : رمز قعر المكس.
- Δ : تابع انتقال معرف بالشكل التالي: $\Delta: Q \times \Sigma^* \times \Gamma^* \rightarrow Q \times \Gamma^*$

تعريف: تكون اللغة مقبولة من أوتومات ذات مكس إذا كانت مجموعة السلاسل التي تنتمي لهذه اللغة تنتقل عند بدايتها من حالتها: مكس فارغ يحوي رمز قعر المكس وحالة الأوتومات الابتدائية، لتصل عند نهايتها (وبعد قراءة كافة رموزها) إلى إحدى حالتين:

- حالة نهائية من حالات الأوتومات
- أو عودة إلى حالة مكس فارغ

مثال: لنعرّف الأوتومات ذات المكس التي تمثل $a^n b^n$ من أجل n صحيح موجب. تكون الأوتومات مؤلفة من العناصر التالية:

$$\left\{ \begin{array}{l} Q = \{s, p, q\} \\ q_0 = s \\ F = \{q\} \\ \Sigma = \{a, b\} \\ \Gamma = \{A, \$\} \\ \tau = \$ \end{array} \right\}, \Delta \left\{ \begin{array}{l} (s, a, \varepsilon) \rightarrow (s, A) \\ (s, b, A) \rightarrow (p, \varepsilon) \\ (p, b, A) \rightarrow (p, \varepsilon) \\ (s, \varepsilon, \$) \rightarrow (q, \varepsilon) \\ (p, \varepsilon, \$) \rightarrow (p, \varepsilon) \end{array} \right.$$

لتكن لدينا الكلمة: aaabbb، تكون حركة الأوتومات الناجمة عن قراءة الكلمة وقبولها ممثلة على النحو التالي:

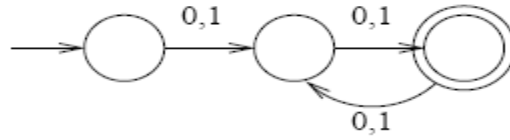
Stack	Input	State	Transition
\$	aaabbb	s	$(s, a, \varepsilon) \rightarrow (s, A)$
\$A	aabbb	s	$(s, a, \varepsilon) \rightarrow (s, A)$
\$AA	abbb	s	$(s, a, \varepsilon) \rightarrow (s, A)$
\$AAA	bbb	s	$(s, b, A) \rightarrow (p, \varepsilon)$
\$AA	bb	p	$(p, b, A) \rightarrow (p, \varepsilon)$
\$A	b	p	$(p, b, A) \rightarrow (p, \varepsilon)$
\$		p	$(p, \varepsilon, \$) \rightarrow (p, \varepsilon)$
\$		q	ACCEPTED

٦ . تمارين

السؤال الأول:

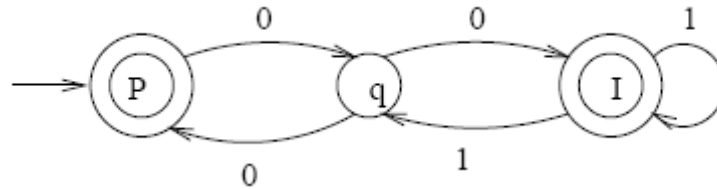
أعط الأوتومات المنتهي الحتمي التي تقبل سلاسل لها طول زوجي (عدا السلسلة الفارغة).

الجواب الأول:

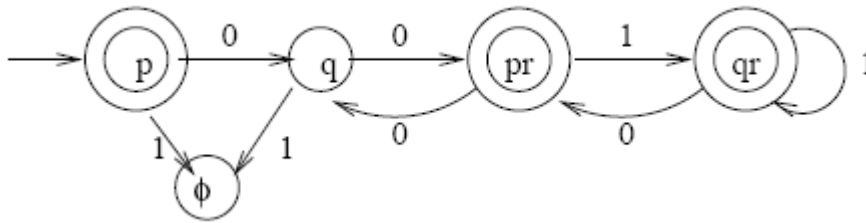


السؤال الثاني:

أعط الأوتومات المنتهي الحتمي المكافئ للأوتومات المنتهي الاحتمالي التالي:



الجواب الثاني:

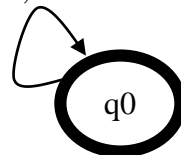


السؤال الثالث:

هل اللغة a^*b^* هي لغة منتظمة؟

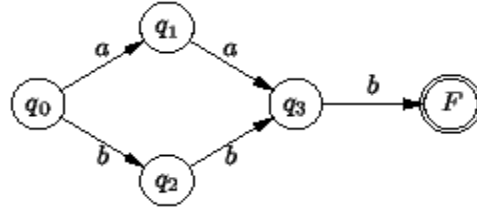
الجواب الثالث:

نعم لأن بإمكاننا رسم أوتومات منته حتمي لها وهو



السؤال الرابع:

أعط التعبير المنتظم المعبر عن الأوتومات المنتهي الحتمي التالي:



الجواب الرابع:

$(aa|bb)b$

الفصل السادس

التحليل القواعدي

تمتلك كل لغة من لغات البرمجة مجموعة من القواعد التي تحدد شكل وبنية الجمل الصحيحة التي يمكن كتابتها باستخدام هذه اللغة. فعلى سبيل المثال، يكون برنامج مكتوب بلغة C مؤلفاً من مقاطع (Blocs) كل مقطع منها مؤلف من مجموعة تعليمات (Instructions) وهكذا دواليك، وذلك وفقاً لقواعد ندعوها القواعد الصرفية (Syntax Rules)، وبحيث تشكل مجموعة القواعد الصرفية، نحواً صرفياً (Grammar).

يتلقى المحلل القواعدي عادةً سلسلةً من الكلمات التي قام المحلل المفرداتي بتمييزها. يعمل المحلل القواعدي على التحقق من أن السلسلة (الجملة) التي تشكلت لديه متوافقة مع النحو الصرفي الخاص بلغة البرمجة التي يُفترض أن النص مكتوب بها. تصبح المسألة على النحو التالي:

- بفرض أن لدينا نحواً صرفياً G.
- بفرض أن لدينا سلسلة m من المفردات.

هل تنتمي m إلى اللغة التي يولدها G؟

يعتمد التحليل الصرفي الذي نتناوله في هذا الفصل على بناء شجرة الاشتقاق (Derivation Tree). إلا أن بناء هذه الشجرة يتم عادةً وفق طريقتين للمسح والتحليل: تحليل نازل (Top-Down Parsing) وتحليل صاعد (Bottom-Up Parsing).

1. النحو الصرفي ومفهوم شجرة الاشتقاق

سنطرح السؤال التالي: في حال كان لدينا لغة ما، كيف يمكن توصيف جميع السلاسل المقبولة التي تنتمي إليها؟ لقد سبق وطرحنا هذا السؤال وأجبنا عليه في فصول سابقة عندما تعاملنا مع التعابير المنتظمة واعتبرناها وسيلة للتعبير. إلا أننا وجدنا أيضاً أن التعابير المنتظمة تصلح للتعبير عن لغات منتظمة، وأن هذه اللغات لا تشكل إلا طيفاً ضيقاً جداً من اللغات التي يمكن أن نتعامل معها. ووجدنا أيضاً أن هناك لغات أكثر اتساعاً (كاللغة $a^n b^n$) لا يمكن التعبير عنها بتعبير منتظم. لذا نحتاج في هذه الحالة إلى أداة أكثر قوة للتعبير عن مثل هذه اللغات. تتمثل هذه الأداة بما ندعوه النحو الصرفي.

a. النحو الصرفي

تمتلك الجملة في أي لغة، بنية محددة بقواعد اللغة، كما هو الحال في اللغة العربية حيث يمكننا القول (مع التبسيط) أن الجملة الفعلية تتألف من فعل يليه فاعل. ويمكن أن نكتب ما سبق على الشكل التالي:

جملة فعلية = فعل فاعل

ندعو ما سبق قاعدة صرفية وتكون هذه القاعدة عادةً، جزءاً من مجموعة القواعد التي تضبط اللغة والتي ندعوها النحو الصرفي.

لنفترض الآن أن بإمكان الفعل و الفاعل أن يأخذا عدة قيم كما هو الحال فيما يلي:

فعل = أكل | شرب
فاعل = خالد | عمر

تمكننا هذه القيم من تشكيل أربع جمل عربية صحيحة من اللغة:

أكل خالد | شرب خالد | أكل عمر | شرب عمر

بالتالي، نستطيع اعتباراً من القواعد الصرفية للغة، اشتقاق الجمل التي تمثلها هذه القواعد. عموماً، تحتوي أية لغة على عدد لانتهائي من الجمل ولكنها لا تحتاج إلى عدد لانتهائي من القواعد. فعلى سبيل المثال، يمكننا لتوليد أي عدد صحيح باستخدام القاعدة التالية:

Number = Digit | Digit Number
Digit = 0|1|2|3|4|5|6|7|8|9

وعليه، يمكننا تعريف النحو الصرفي والقواعد الصرفية التي تنتمي إليه على النحو التالي:

تعريف: نعرف النحو الصرفي (Grammar) بأنه رباعية $G = (V_T, V_N, S, P)$ حيث:
 V_T : مجموعة غير فارغة من رموز الأبجدية (Terminals) التي ندعوها الرموز الأولية.
 V_N : مجموعة غير فارغة من الرموز الوسيطة (Non Terminals) حيث $V_T \cap V_N = \emptyset$.
 S : رمز البداية ويكون $S \in V_N$.
 P : مجموعة القواعد الصرفية الخاصة بالنحو G .

تعريف: تحدد القاعدة الصرفية $\alpha \rightarrow \beta$ أنه يمكننا استبدال سلسلة الرموز α (حيث $\alpha \in (V_T \cup V_N)^+$) بسلسلة الرموز β (حيث $\beta \in (V_T \cup V_N)^+$). ندعو α الجزء اليساري من القاعدة، في حين ندعو β الجزء اليميني منها.

لنأخذ مثلاً النحو الصرفي التالي:

$$G \begin{cases} V_T = \{a, b\} \\ V_N = \{A\} \\ S : A \\ P : A \rightarrow aAb \mid \varepsilon \end{cases}$$

يمثل هذا النحو الصرفي اللغة التي سبق وتطرقتنا إليها وهي $(a^n b^n)$ وهو ما سنستعرضه في الفقرة اللاحقة.

b. شجرة الاشتقاق

تعريف: ندعو "اشتقاقاً" (Derivation) التطبيق المتتالي لمجموعة من القواعد اعتباراً من كلمة تنتمي إلى $(V_T \cup V_N)^+$

نستخدم إشارة السهم (\rightarrow) للإشارة إلى اشتقاق ناجم عن تطبيق قاعدة واحدة، كما نستخدم إشارة السهم (\rightarrow^*) المزود بنجمة للإشارة إلى اشتقاق ناتج عن تطبيق عدد n من القواعد حيث $n \geq 0$.
أمثلة:

$$G \left\{ \begin{array}{l} V_T = \{a, b\} \\ V_N = \{A\} \\ S : A \\ P : A \rightarrow aAb \mid \varepsilon \end{array} \right. \rightarrow S \rightarrow aAb \rightarrow aaAbb \rightarrow aaaAbbb \rightarrow aaabbb \rightarrow S^* \rightarrow aaabbb$$

$$\begin{array}{l} \text{Number} = \text{Digit} \mid \text{Digit Number} \\ \text{Digit} = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{array} \rightarrow \text{Number} \rightarrow 5 \text{Number} \rightarrow 52 \rightarrow \text{Number}^* \rightarrow 52$$

تعريف: من أجل نحو صرفي $G = (V_T, V_N, S, P)$ ، نعرف اللغة التي يولدها هذا النحو، كما يلي:

$$L(G) = \{w \in (V_T)^* : S \rightarrow^* w\}$$

تعريف: من أجل نحو صرفي $G = (V_T, V_N, S, P)$ ، ندعو شجرة الاشتقاق الشجرة التي يكون جذرها هو الرمز S .

- ❖ أوراقها هي مجموعة الرموز الأولية المنتمية إلى V_T أو الرمز ε .
- ❖ عقدها هي مجموعة الرموز الوسيطة المنتمية إلى V_N .
- ❖ تكون العقد التالية (العقد الأبناء) لعقدة α هي مجموعة العقد β_1, \dots, β_n إذا فقط إذا كان:
$$\alpha \rightarrow \beta_1, \dots, \beta_n$$

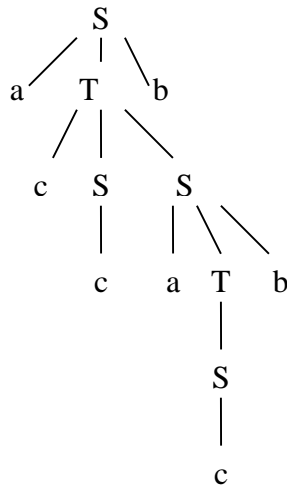
يكون **الاشتقاق يسارياً** (Leftmost Derivation) إذا استبدلنا عند تنفيذ كل عملية اشتقاق للجملة اليمينية، الرمز الوسيط (non terminal symbol) الموجود على أقصى يسار هذه الجملة بالقاعدة المناسبة له.
يكون **الاشتقاق يمينياً** (Rightmost Derivation) إذا استبدلنا، عند تنفيذ كل عملية اشتقاق للجملة اليمينية، الرمز الوسيط (non terminal symbol) الموجود على أقصى يمين هذه الجملة بالقاعدة المناسبة له.
مثال:

ليكن لدينا النحو الصرفي التالي:

$$G = (\{a, b, c\}, \{S, T\}, S, P), P : \begin{cases} S \rightarrow aTb \mid c \\ T \rightarrow cSS \mid S \end{cases}$$

تكون شجرة الاشتقاق (الناجمة عن الاشتقاق اليميني أو اليساري) للكلمة (accacbb) هي الشجرة التالية:

$$\begin{array}{l} \text{Leftmost} : S \rightarrow aTb \rightarrow acSSb \rightarrow accSb \rightarrow accaTbb \rightarrow accaSbb \rightarrow accacbb \\ \text{Rightmost} : S \rightarrow aTb \rightarrow acSSb \rightarrow acSaTbb \rightarrow acSaSbb \rightarrow acSacbb \rightarrow accacbb \end{array}$$



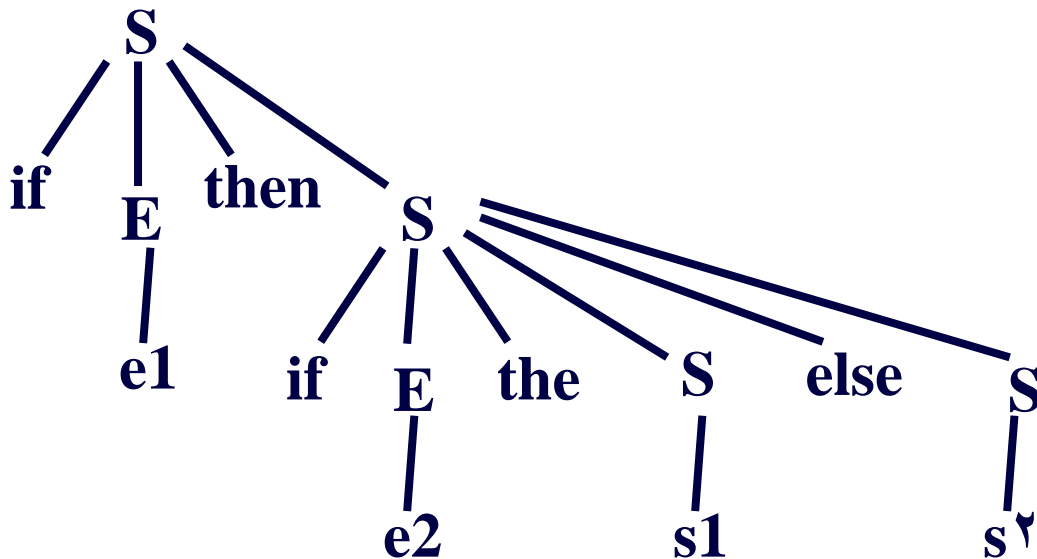
قد تتسبب القواعد الصرفية الموضوعية بتوليد أكثر من شجرة اشتقاق يميني أو أكثر من شجرة اشتقاق يساري ندعو هذه المشكلة بمشكلة الغموض (Ambiguity). فعلى سبيل المثال، لنأخذ قواعد الجملة الشرطية التالية:

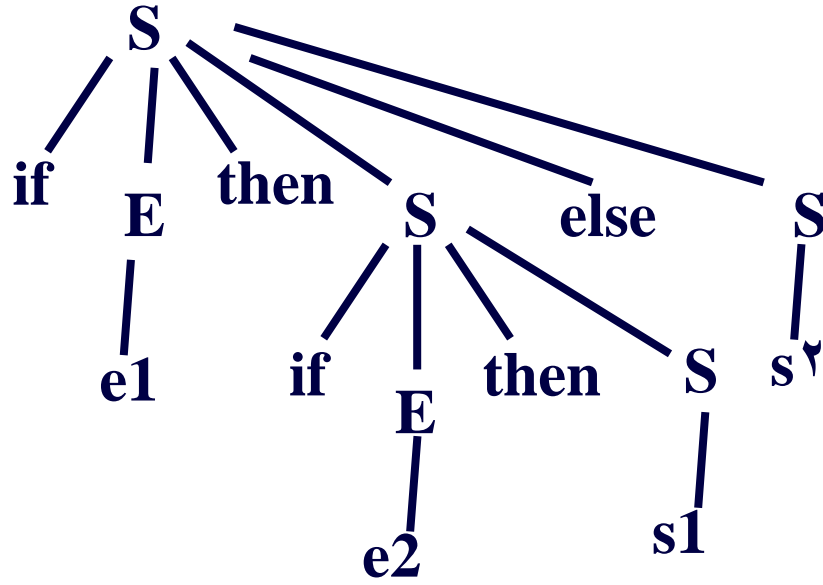
$S \rightarrow \text{if } E \text{ then } S$ $ \text{if } E \text{ then } S \text{ else } S$

في حال قمنا باشتقاق الجملة:

$\text{if } e1 \text{ then } s1 \text{ if } e2 \text{ then } s2 \text{ else } s3$

فسنجد أننا، في حالة الاشتقاق اليساري، سنحصل على شجرتين مختلفتين:





مما يعني أن القواعد تعاني من حالة غموض وأن اشتقاقها اليساري (أو اليميني) سيؤدي إلى الوصول إلى حالة يصعب فيها التنبؤ بكيفية إتمام عملية الاشتقاق. ومن الواضح هنا هو أن الغموض ناجم عن عدم تحديد القواعد التي تضبط لتبعية تعليمة else إلى تعليمة if في حال وجود عدة بنى لتعليمة if متداخلة ببعضها البعض.

٢ . تنفيذ التحليل القواعدي

يستقبل المحلل القواعدي سلسلة من المفردات (رموز نهائية) من المحلل المفرداتي. وتكون مهمته تحديد صحتي هذه الجملة وفقاً للقواعد الصرفية التي يعمل بها. لذا يعمل المحلل القواعدي على بناء شجرة اشتقاق جملة اعتباراً من تلك القواعد الصرفية، ويعمل على التأكد من عدم وجود أي حالات غموض فيها. يعتمد المحلل الصرفي في تحليله للجملة على إحدى مقاربتين: تحليل نازل يبدأ من رمز البداية S للقواعد ويسعى بتطبيقه للاشتقاق الوصول إلى الجملة المطلوب التحقق منها. وآخر صاعد يبدأ من الجملة ويسعى لتعويض سلاسل الكلمات المتشكلة من الرموز النهائية، بالرموز الوسيطة التي تعبر عنها وفقاً للقواعد، تمهيداً للوصول إلى رمز البداية S.

٣ . تحليل نازل من الأعلى إلى الأسفل

المبدأ: بناء شجرة اشتقاق الجملة التي نقوم بالتأكد من صحتها، بشكل نازل يبدأ من رمز البداية S للقواعد ويسعى من خلال تطبيقه للاشتقاق، إلى الوصول للجملة المطلوبة.

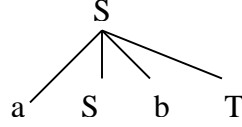
a. أمثلة

المثال الأول: لتكن لدينا القواعد التالية، حيث يعبر S عن رمز البداية، وحيث T رمز وسيط:

$$\begin{array}{l} S \rightarrow aSbT \mid cT \mid d \\ T \rightarrow aT \mid bS \mid c \end{array}$$

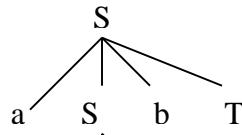
نريد تحليل الكلمة: $w=acbbadbc$. لنقلع من جذر الشجرة وهو الرمز S . تؤدي قراءة الكلمة a إلى التقدم في عملية بناء الشجرة لنحصل على:

$W=\underline{a}ccbbadba$

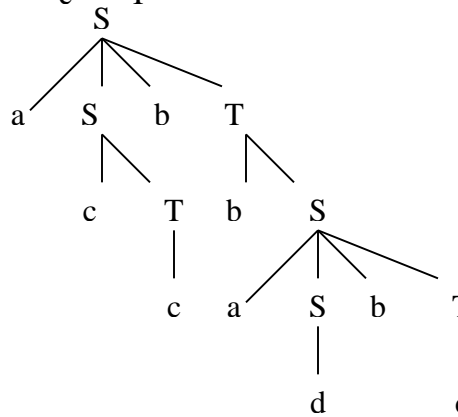


تؤدي قراءة الكلمة الثانية c إلى التقدم خطوة أخرى:

$W=\underline{ac}cbbadba$



وهكذا حتى نصل إلى:



طالما أننا وجدنا شجرة اشتقاق، فهذا يعني أن الجملة صحيحة قواعدياً وفق القواعد المحددة في بداية المثال.

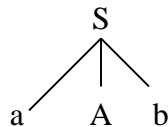
المثال الثاني: لتكن لدينا القواعد التالية، حيث يعبر S عن رمز البداية، وحيث A رمز وسيط:

$$\begin{array}{l} S \rightarrow aAb \\ A \rightarrow cd | c \end{array}$$

نريد تحليل الكلمة: $w=acb$

لنقلع من جذر الشجرة وهو الرمز S ، فتؤدي قراءة الكلمة a إلى التقدم في عملية بناء الشجرة لنحصل على:

$W=\underline{a}cb$



ولكن عند قراءة c لن نستطيع تحديد فيما إذا كان علينا أخذ القاعدة $(A \rightarrow c d)$ أو القاعدة $(A \rightarrow c)$. لتحديد الأمر، يتوجب علينا قراءة الكلمة التالية، وهي (b) ، وإعطاء إمكانية الرجوع إلى الخلف بحيث نجرب القاعدة الأولى $(A \rightarrow c d)$ ونتابع، وفي حال لم يجر الأمر بالشكل الصحيح نعود إلى الخلف، ونعيد بناء هذا الجزء مع القاعدة الثانية. تؤدي العودة إلى الخلف (backtracking) إلى ارتفاع كلفة عملية التحليل.

المثال الثالث: لتكن لدينا القواعد التالية، حيث يعبر S عن رمز البداية:

$$S \rightarrow aSb \mid aSc \mid d$$

نريد تحليل الكلمة: $w = aaaaaaadbbcbbbc$

من الواضح هنا أننا نحتاج منذ البداية لتحديد القاعدة التي يجب الانطلاق منها. للأسف لن نستطيع تحديدها إلا في نهاية الاشتقاق مما سيجعلنا نبني ثم نعود إلى الوراء عدة مرات قبل اكتشاف الشجرة الصحيحة.

المثال الرابع: لتكن لدينا القواعد التالية التي توصف تعابير حسابية، حيث يعبر E عن رمز البداية، وحيث F و

T رموز وسيط:

$$\begin{array}{l} E \rightarrow TE' \\ E' \rightarrow +TE' \mid \varepsilon \\ T \rightarrow FT' \\ T' \rightarrow *FT' \mid \varepsilon \\ F \rightarrow (E) \mid Id \end{array}$$

من الواضح هنا أننا لا نستطيع المتابعة بالأسلوب التجريبي السابق، وبأننا بحاجة لمنهجية تساعدنا في تنفيذ عملية التحليل.

يمكننا أن نلخص منهجية التحليل النازل التي نحتاجها بما يلي:

نحتاج إلى جدول يحدد لنا، في حال قمنا بقراءة كلمة ما، وفي حال كنا في مرحلة اشتقاق معينة عند عقدة معينة للشجرة ممثلة برمز وسيط، ما هي القاعدة الخاصة بهذا الرمز التي يجب اختيارها لمتابعة بناء الشجرة.

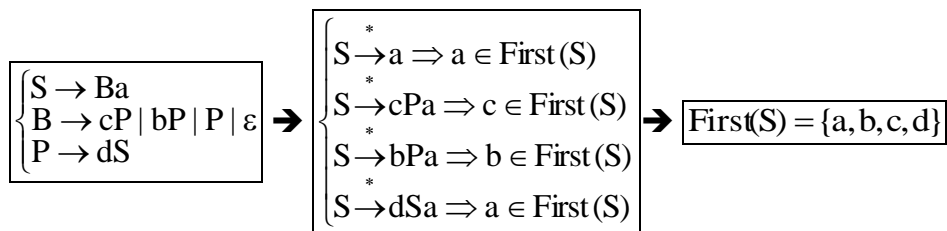
b. التحليل LL(1)

في بداية هذا التحليل، نحتاج لحساب مجموعتين: مجموعة الرموز الأولى (First)، ومجموعة الرموز اللاحقة (Follow)، ومن ثم سنعتمد على هاتين المجموعتين في بناء الجدول المطلوب.

مجموعة الرموز الأولى (First):

من أجل كل سلسلة α مؤلفة من رموز أولية ورموز وسيطة، نبحث عن $First(\alpha)$ وهي مجموعة الرموز الأولية التي تبدأ بها السلسلة المشتقة من α . بمعنى آخر نبحث عن الرموز الأولية a بحيث: $\alpha \rightarrow a\beta^*$ حيث $a\beta^*$ سلسلة مؤلفة من رموز أولية ووسيط.

مثال:



وتكون خوارزمية بناء المجموعة First على النحو التالي:

Repeat

if (X is non terminal and $X \rightarrow Y_1 Y_2 \dots Y_n$ is a production where Y_i is either terminal or non terminal) **then**

{

Add elements of (First(Y_1)-to First(X);) { ε }

if ($\exists j \in [2..n]$ where for each $i \in [1..j-1]$ we have $\varepsilon \in \text{First}(Y_i)$) **then**

First(X) = First(X) \cup (First(Y_j) - { ε })

for $i=1..n$ **do if** $\varepsilon \in \text{First}(Y_i)$ **then** First(X) = First(X) \cup { ε }

}

if (X is non terminal and we have the production $X \rightarrow \varepsilon$) **then**

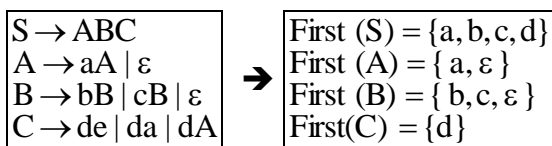
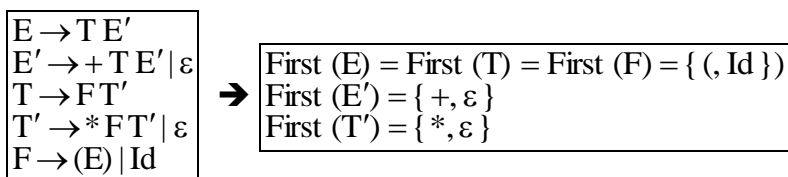
First(X) = First(X) \cup { ε }

if (X is a terminal) **then**

First(X) = {X}

until First(X) has no changes

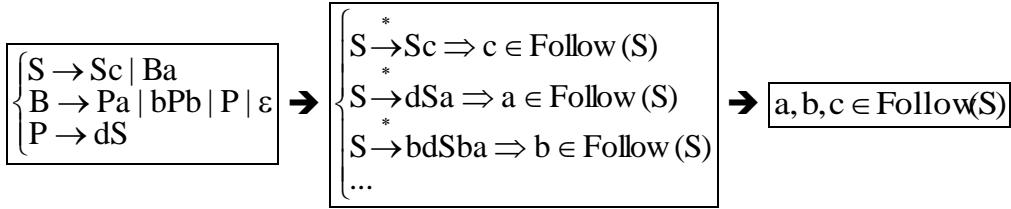
أمثلة:



مجموعة الرموز اللاحقة (Follow):

من أجل كل رمز وسيط A، نبحث عن مجموعة الرموز اللاحقة Follow(A)، وهي مجموعة الرموز الأولية التي يمكن أن تظهر مباشرة على يمين الرمز A في أي اشتقاق $S \xrightarrow{*} \alpha A \beta$.

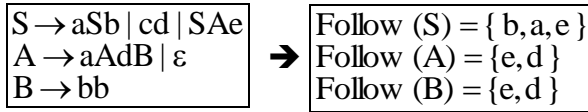
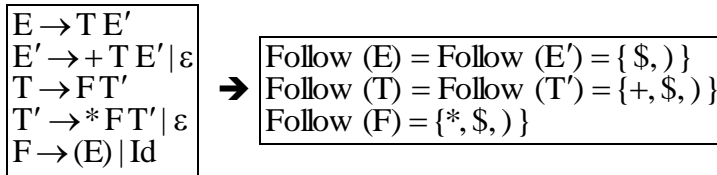
مثال:



وتكون خوارزمية بناء المجموعة Follow على النحو التالي:

Repeat
 $\text{Follow}(S) = \text{Follow}(S) \cup \{\$\}$ (where \$ is the end symbol, S is the starting Symbol)
if $(\exists A \rightarrow \alpha B \beta$ where B is non terminal) **then**
 $\text{Follow}(B) = \text{Follow}(B) \cup (\text{First}(\beta) - \{\varepsilon\})$
if $(\exists A \rightarrow \alpha B$ where B is non terminal) **then**
 $\text{Follow}(B) = \text{Follow}(B) \cup \text{Follow}(A)$
if $(\exists A \rightarrow \alpha B \beta$ where B is non terminal and $\varepsilon \in \text{First}(\beta)$) **then**
 $\text{Follow}(B) = \text{Follow}(B) \cup \text{Follow}(A)$
Until the sets Follow has no changes

أمثلة:

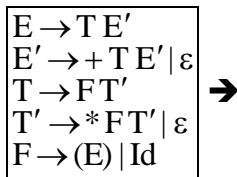


بناء الجدول:

لبناء جدول القواعد M الذي سنستخدمه في عملية التحليل، نعتمد على الخوارزمية التالية:

for each production $X \rightarrow Y$ **do**
{
for each $(a \neq \varepsilon)$ in $\text{First}(X)$ **do** $M[X, a] = X \rightarrow Y$
if $(X \rightarrow \varepsilon)$ **then for** each b in $\text{Follow}(X)$ **do** $M[X, b] = X \rightarrow \varepsilon$
}

مثال:



	Id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow Id$			$F \rightarrow (E)$		

تنفيذ التحليل:

- لنفرض أن لدينا جملة نريد تحليلها بأسلوب التحليل النازل. سنتعامل خوارزمية التحليل مع ثلاثة مكونات:
- الشريط Word الذي سيحوي الجملة (التي تنتهي بالمحرف \$) وبحيث يكون للشريط مؤشر wp على الحرف المقروء؛
 - مكس Stack له قعر (سنفترضه ممثل بالمحرف \$) له مؤشر sp إلى قمته ويتعامل مع إجرائيتين Pop تحذف أعلى القمة وتهبط بالمؤشر إلى القمة الجديدة، وإجرائية Push(c) التي تضيف العنصر c إلى أعلى المكس وترفع المؤشر ليشير إليه؛
 - جدول القواعد الذي سبق وقمنا ببنائه في الفقرة السابقة.

تكون خوارزمية تحليل جملة W على الشكل التالي:

```

Repeat
{
Stack[sp]=S;
X=Stack[sp];
a=W[wp];
if (X is non terminal) then
{
    if (M[X, a]= " X → Y1...Yn ") then
    {
        Pop;
        for (i=n down to 1) do
            Push( Yi )
    }
    else ERROR;
}
else
{
    if (X= =$) then
        if (a = = $) then ACCEPTED
        else ERROR;
    else
        if (X = = a) then { Pop; wp=wp+1; a=W[wp]; }
        else ERROR;
}
Until ERROR or ACCEPTED

```

مثال:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \epsilon$
$F \rightarrow (E) \mid Id$

لنأخذ الجدول السابق الذي سبق وبنينا للقواعد:

	Id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow Id$			$F \rightarrow (E)$		

لنحلل الجملة $a+b*c\$$ حيث a و b و c معرفات (Identifiers):

Stack	Input Word	Rule
\$E	a+b*c\$	$E \rightarrow TE'$
\$ET	a+b*c\$	$T \rightarrow FT'$
\$ETF	a+b*c\$	$F \rightarrow Id$
\$ETa	a+b*c\$	
\$ET	+b*c\$	$T' \rightarrow \epsilon$
\$E	+b*c\$	$E' \rightarrow +TE'$
\$ET+	+b*c\$	
\$ET	b*c\$	$T \rightarrow FT'$
\$ETF	b*c\$	$F \rightarrow Id$
\$ETb	b*c\$	
\$ET	*c\$	$T' \rightarrow *FT'$
\$ETF*	*c\$	
\$ETF	c\$	$F \rightarrow Id$
\$Etc	c\$	
\$ET	\$	$T' \rightarrow \epsilon$
\$E	\$	$E' \rightarrow \epsilon$
\$	\$	ACCEPTED

c. النحو LL(1)

لا يمكن تطبيق الخوارزمية السابقة على جميع أنواع النحو الصرفي. فإذا وجد في إحدى خانات جدول القواعد عدد من القواعد عوضاً عن قاعدة وحيدة، لن تتمكن الخوارزمية من تحديد أي قاعدة هي القاعدة الصحيحة والتي يجب اعتمادها.

لذا، نعرّف نحو صرفي بأنه "نحو صرفي من النمط LL(1)" في حال كان جدول القواعد المبني لهذا النحو لا يحوي على خانات فيها عدة قواعد متعددة. ونعني بالمصطلح LL(1):
 (المسح من اليسار ومن ثم الاشتقاق اليساري للقواعد مع توقع حرف وحيد إلى الأمام)
 (Left scanning Leftmost derivation with 1 look ahead symbol)

مثال:

النحو التالي ليس نحو من النمط LL(1)

$$\begin{array}{l} S \rightarrow \text{if } E \text{ then } SS' \mid b \\ S' \rightarrow \text{else } S \mid \varepsilon \\ E \rightarrow b \end{array}$$
 \rightarrow

$$\begin{array}{l} \text{First } (S) = \{\text{if}, b\} \\ \text{First } (S') = \{\text{else}, \varepsilon\} \\ \text{First } (E) = \{b\} \\ \text{Follow } (S) = \{\$\} \\ \text{Follow } (S') = \{\text{else}, \$\} \\ \text{Follow } (E) = \{\text{then}, \$\} \end{array}$$
 \rightarrow

	b	else	if	then	\$
S	$S \rightarrow b$		$S \rightarrow \text{if } E \text{ then } SS' \mid b$		
S'		$S' \rightarrow \varepsilon$ $S' \rightarrow \text{else } S$			$S' \rightarrow \varepsilon$
E	$E \rightarrow b$				

نتيجة: أي نحو صرفي يتصف بإحدى الصفتين التاليتين:

- i. عودي يساري (left recursive).
 - ii. له معاملات يسارية مشتركة ولكنه غير مختصر عبر حساب معاملاته اليسارية المشتركة (not left factorized).
- لا يكون من النمط LL(1).

d. العودية اليسارية (Left Recursion)

تعريف: يكون النحو الصرفي "عودي يساري مباشر" إذا كان فيه رمز وسيط A له قاعدة صرفية من الشكل:

$$A \rightarrow A\alpha : \alpha \in (V \cup T)^*$$

مثال:

$$\begin{array}{l} S \rightarrow ScA \mid B \\ A \rightarrow Aa \mid \varepsilon \\ B \rightarrow Bb \mid d \mid \varepsilon \end{array}$$

نظرية:

لحذف العودية اليسارية المباشرة نطبق الخوارزمية التالية:

$$\forall A : (A \rightarrow Aa \mid \beta) \Leftrightarrow \begin{cases} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' \mid \varepsilon \end{cases}$$

ويكون النحو الناتج مكافئ تماماً للنحو الأصلي حيث يولد نفس اللغة التي يولدها النحو الأصلي

مثال: بتطبيق الخوارزمية السابقة نحصل على:

$$\begin{array}{l} S \rightarrow ScA \mid B \\ A \rightarrow Aa \mid \varepsilon \\ B \rightarrow Bb \mid d \mid e \end{array} \rightarrow \begin{array}{l} S \rightarrow BS' \\ S' \rightarrow cAS' \mid \varepsilon \\ A \rightarrow A' \\ A' \rightarrow aA' \mid \varepsilon \\ B \rightarrow dB' \mid eB' \\ B' \rightarrow bB' \mid \varepsilon \end{array}$$

تعريف: يكون النحو الصرفي "عودي يساري" إذا كان فيه رمز وسيط A له اشتقاق من الشكل:

$$A \xrightarrow{+} A\alpha : \alpha \in (V \cup T)^*$$

مثال: هذا النحو عودى يساري بسبب الرمز الوسيط S وقاعدته، ولكنه ليس عودياً يسارياً مباشراً:

$$\begin{array}{l} S \rightarrow Aa \mid b \\ A \rightarrow Ac \mid Sd \mid \varepsilon \end{array} \rightarrow S \rightarrow Aa \rightarrow Sda$$

نظرية:

لحذف العودية اليسارية نطبق الخوارزمية التالية:

Make an order between non terminals: A_1, \dots, A_n

for $i=1$ to n **do**

{

for $j=1$ to $i-1$ **do**

{

Substitute each production of the form $A_i \rightarrow A_j\alpha$ or

$A_j \rightarrow \beta_1 \mid \dots \mid \beta_p$ by a production of the form $A_i \rightarrow \beta_1\alpha \mid \dots \mid \beta_p\alpha$

}

Eliminate the direct left recursion of productions of A_i

}

ويكون النحو الناتج مكافئ تماماً للنحو الأصلي حيث يولد نفس اللغة التي يولدها النحو الأصلي.

مثال ١:

$$\begin{array}{l} S \rightarrow Aa \mid b \\ A \rightarrow Ac \mid Sd \mid \varepsilon \end{array} \rightarrow \begin{array}{l} S \rightarrow Aa \mid b \\ A \rightarrow bdA' \mid A' \\ A' \rightarrow cA' \mid adA' \mid \varepsilon \end{array}$$

مثال ٢: لنأخذ المثال التالي

$$\begin{array}{l} S \rightarrow Sa \mid TSc \mid d \\ T \rightarrow TbT \mid \varepsilon \end{array} \rightarrow \begin{array}{l} S \rightarrow TScS' \mid dS' \\ S' \rightarrow aS' \mid \varepsilon \\ T \rightarrow T' \\ T' \rightarrow bTT' \mid \varepsilon \end{array}$$

لكننا نلاحظ هنا أنه وبعد تطبيق الخوارزمية يمكن أن نحصل على عودية يسارية غير مباشرة:

$$S \rightarrow TScS' \rightarrow T'ScS' \rightarrow ScS'$$

يعود السبب في ذلك إلى وجود قاعدة $T' \rightarrow \varepsilon$,

نتيجة: الخوارزمية لا تعمل بشكل صحيح إذا وجدت قواعد من النمط $A \rightarrow \varepsilon$.

e. حساب المعاملات اليسارية المشتركة

عندما يكون على خوارزمية التحليل اتخاذ قرار بين عدد من القواعد، تساعد عملية حساب المعاملات اليسارية المشتركة في تأخير اتخاذ القرار ريثما تكون الجملة قد أصبحت أكثر اكتمالاً مما يسمح باتخاذ قرار أفضل.

مثال: لتكن لدينا القواعد الصرفية التالية:

$$\begin{array}{l} S \rightarrow bacdAbd \mid bacdBcca \\ A \rightarrow aD \\ B \rightarrow cE \\ C \rightarrow \dots \\ E \rightarrow \dots \end{array}$$

في هذه الحالة، نجد أن لدينا خيارين من أجل S، ولمعرفة الخيار الصائب، يتوجب علينا قراءة ٤ رموز والوصول إلى الرمز الخامس، لنعرف إذا كنا سنحصل على a (حالة الرمز الوسيط A) أو سنحصل على c (حالة الرمز الوسيط B). إذاً لا يمكن لنا منذ البداية معرفة القاعدة التي يتوجب علينا اختيارها وهو ما يتعارض مع مفهوم النحو الصرفي من النمط LL(1).

نظرية: يمكن حساب المعاملات المشتركة (في حال وجدت) في نحو الصرفي اعتماداً على الخوارزمية التالية:

for each non terminal A do
 {
 Find the common prefix $\alpha \neq \varepsilon$ and substitute: $A \rightarrow \alpha B_1 \mid \dots \mid \alpha B_n \mid \gamma_1 \mid \dots \mid \gamma_p$
 by the two following rules:
 $\left\{ \begin{array}{l} A \rightarrow \alpha A' \mid \gamma_1 \mid \dots \mid \gamma_p \\ A' \rightarrow B_1 \mid \dots \mid B_n \end{array} \right.$
 }

مثال:

$$\begin{array}{l} S \rightarrow aEbS \mid aEbSeB \mid a \\ E \rightarrow bcB \mid bca \\ B \rightarrow ba \end{array} \rightarrow \begin{array}{l} S \rightarrow aEbSS' \mid a \\ S' \rightarrow eB \mid \varepsilon \\ E \rightarrow bcE' \\ E' \rightarrow B \mid a \\ B \rightarrow ba \end{array}$$

f. النحو النظيف

نقول عن نحو صرفي أنه "نحو نظيف" إذا كان خالياً من قواعد من الشكل: $A \rightarrow \varepsilon$. يمكن تحويل نحو صرفي إلى نحو صرفي نظيف إذا قمنا من أجل كل رمز A يمتلك قاعدة من الشكل $A \rightarrow \varepsilon$ ، باستبدال كل ظهور له ضمن الجهة اليمينية من أي قاعدة من القواعد الصرفية بالرمز ε .

مثال:

$$\begin{array}{|l} S \rightarrow aTb | ab | aU \\ T \rightarrow bTaTA | baTA | bTaA | baA \\ U \rightarrow aU | b \end{array} \rightarrow \begin{array}{|l} S \rightarrow aTb | aU \\ T \rightarrow bTaTA | \varepsilon \\ U \rightarrow aU | b \end{array}$$

g. استنتاج

يمكن أتمتة عملية التحليل النازل، لنحو صرفي من النوع LL(1) على أن يكون النحو:

١. غير غامض وهو أمر يعود إلى مصمم النحو.
٢. لا يحوي على عودية يسارية بحيث تتم إزالة العودية اليسارية (إذا وجدت) باستخدام خوارزمية التحويل إلى نحو دون عودية يسارية.
٣. مختصر بعد حساب معاملاته اليسارية المشتركة (إذا وجدت) وذلك باستخدام خوارزمية التحويل التي سبق وذكرناها.
٤. بناء جدول القواعد بعد حساب مجموعتي الرموز الأولى والرموز اللاحقة (First & Follow) وفق خوارزميات الحساب المناسبة.
٥. تنفيذ عملية التحليل باستخدام المكس وفق خوارزمية التحليل النازل.

مثال:

$$1. \text{ اعتباراً من نحو غامض: } E \rightarrow E + E | E * E | (E) | Id$$

$$2. \text{ يجب هيكلته باستخدام الخواص التجميعية للعمليات الحسابية ليصبح: } \begin{cases} E \rightarrow E + T | T \\ T \rightarrow T * F | F \\ F \rightarrow (E) | nb \end{cases}$$

$$3. \text{ يتم تطبيق خوارزمية إلغاء العودية اليسارية لنحصل على: } \begin{cases} E \rightarrow TE' \\ E' \rightarrow +TE' | \varepsilon \\ T \rightarrow FT' \\ T' \rightarrow *FT' | \varepsilon \\ F \rightarrow (E) | Id \end{cases}$$

٤. نلاحظ أن هذا النحو هو من النمط LL(1) وليس عودياً من اليسار ولا حاجة لاختصاره بحساب المعاملات اليسارية المشتركة لأنها محسوبة، لذا يمكن تطبيق خوارزمية التحليل مباشرةً عليه.

٤ . تحليل صاعد من الأسفل إلى الأعلى

يعتمد التحليل الصاعد في مبدأ عمله على بناء شجرة الاشتقاق من الأسفل (من الأوراق التي تمثل الرموز الأولية مروراً بالعقد التي تمثل الرموز الوسيطة) إلى الأعلى (رمز البداية).

يدعى النموذج العام المستخدم بنموذج **السحب/الاختصار (Shift/Reduce)** ويقوم هذا النموذج على عمليتين هما:

- .iii **السحب (Shift):** التي تعني سحب المؤشر المار على الجملة المقروءة بمقدار كلمة إلى الأمام.
- .iv **الاختصار (Reduce):** التي تعني اختصار الجملة من خلال تعويض مجموعة من الرموز التي يمكن أن تشكل سلسلة يمينية لقاعدة صرفية برمز وحيد هو الرمز الظاهر في الناحية اليسارية من هذه القاعدة.

مثال: لنأخذ القاعدة: $S \rightarrow aSbSc$ ولنأخذ الجملة: $u=aaacbaacbcbcbcbacbc$

- لنبدأ من الحرف الأول: $aaacbaacbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aaacbaacbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aaacbaacbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aaacbaacbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow c$.
- نختصر فنحصل على: $aaaSbaacbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aaaSbaacbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نتابع عمليات السحب
- نسحب باتجاه الحرف التالي: $aaaSbaacbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow c$.
- نختصر فنحصل على: $aaaSbaaSbcbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aaaSbaaSbcbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aaaSbaaSbcbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow c$.
- نختصر فنحصل على: $aaaSbaaSbcbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow aSbS$.
- نختصر فنحصل على: $aaaSbaaSbcbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نتابع عمليات السحب
- نسحب باتجاه الحرف التالي: $aaaSbaaSbcbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow c$.
- نختصر فنحصل على: $aaaSbaaSbcbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow aSbS$.
- نختصر فنحصل على: $aaaSbaaSbcbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow aSbS$.
- نختصر فنحصل على: $aaSbcbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aaSbcbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aaSbcbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow c$.
- نسحب باتجاه الحرف التالي: $aaSbcbcbcbcbacbc$ حيث يمكن أن نختصر وفق $S \rightarrow aSbS$.
- نختصر فنحصل على: $aSbcbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aSbcbcbcbcbacbc$ حيث لا يمكن أن نختصر فنسحب.

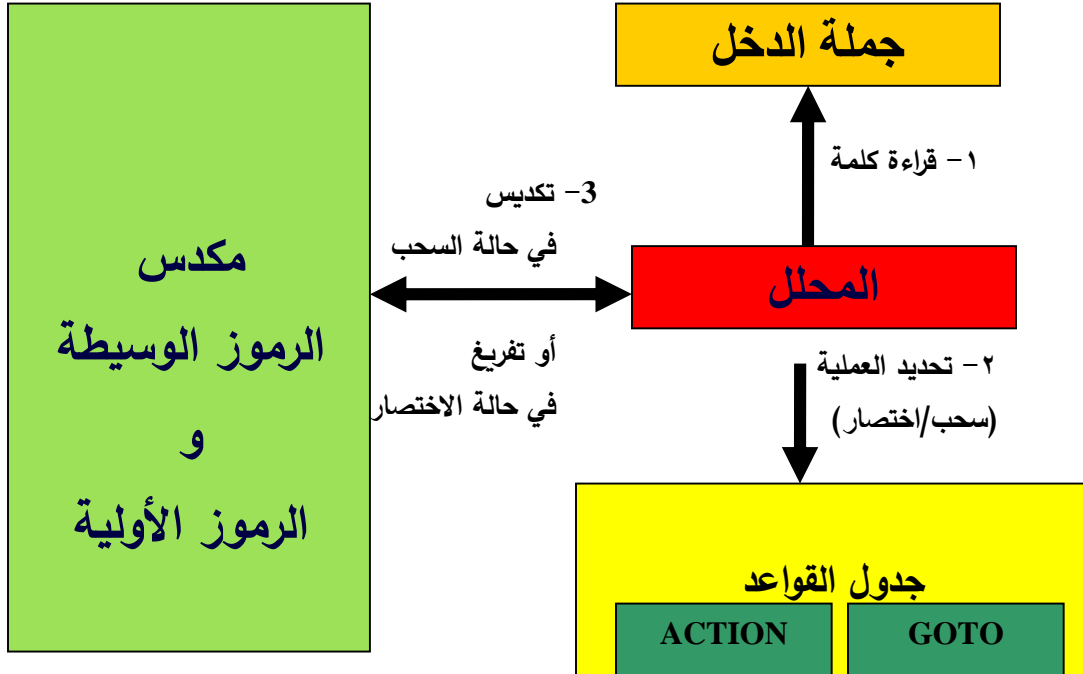
- نسحب باتجاه الحرف التالي: $aSb\underline{a}cbc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aSba\underline{c}bc$ حيث يمكن أن نختصر وفق $S \rightarrow c$.
- نختصر فنحصل على: $aSba\underline{S}bc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aSba\underline{S}bc$ حيث لا يمكن أن نختصر فنسحب.
- نسحب باتجاه الحرف التالي: $aSbaSb\underline{S}$ حيث يمكن أن نختصر وفق $S \rightarrow aSbS$.
- نختصر فنحصل على: $aSb\underline{S}$ حيث يمكن أن نختصر وفق $S \rightarrow aSbS$.

نختصر فنحصل على \underline{S} ونصل إلى رمز البداية، مما يعني أن الجملة التي قمنا بتحليلها صحيحة.

بالنتيجة، سيكون من الضروري الحصول على جدول يساعدنا في معرفة العملية التي يجب تطبيقها، وهل هي عملية سحب أم عملية اختصار.

a. خوارزمية التحليل LR:

ستسمح هذه الطريقة بتحليل الجمل التي تنتمي لنحو صرفي من النمط LR (Leftmost scanning Rightmost derivation). ويكون الجدول في هذه الحالة عبارة عن أوتومات ذات مكس بحيث يسمح لنا في كل مرة نقوم فيها بقراءة حرف، معرفة فيما إذا كانت العملية التي يجب تطبيقها هي عملية سحب او عملية اختصار. تتم عملية التحليل وفق المخطط التالي:



الخوارزمية: سنتعرف على خوارزمية التحليل من خلال مثال. ليكن لدينا النحو الصرفي التالي بعد ترقيم قواعده والجملة التي نريد تحليلها:

$$\begin{cases} (1) E \rightarrow E + T \\ (2) E \rightarrow T \\ (3) T \rightarrow T * F \\ (4) T \rightarrow F \\ (5) F \rightarrow (E) \\ (6) F \rightarrow Id \end{cases}$$

ولنفرض أن جملة الدخل التي نريد تحليلها هي: $[a * b + c \$]$ والتي تنتهي برمز نهاية الجملة $\$$. ولنفرض أن جدول القواعد على النحو التالي (سنتعرف على طريقة بناء الجدول لاحقاً):

	ACTION						GOTO		
	Id	+	*	()	\$	E	T	F
0	S 5			S 4			1	2	3
1		S 6				Acc			
2		R 2	R 7		R 2	R 2			
3		R 4	R 4		R 4	R 4			
4	S 5			S 4			8	2	3
5		R 6	R 6		R 6	R 6			
6	S 5			S 4				9	3
7	S 5			S 4					10
8		S 6			S 11				
9		R 1	S 7		R 1	R 1			
10		R 3	R 3		R 3	R 3			
11		R 5	R 5		R 5	R 5			

- يكون المكس في الحالة الابتدائية حاوياً على رمز قعر المكس $\$$ يليه رمز بداية النحو (E في مثالنا) ويليه رقم الحالة الأولى (0 في مثالنا). يكون مؤشر أعلى المكس مؤشراً على هذه الحالة أي أننا نفترض أننا في الحالة الابتدائية (رقم 0 في حالتنا)
- يكون مؤشر بداية الجملة يُوْشر إلى الكلمة الأولى منها (a في مثالنا).
- يحتوي العمود اليساري للجدول الحالة s_i الحالية. كما يحتوي السطر العلوي رموز النحو الصرفي.
- اعتباراً من الحالة s_i اقرأ الكلمة a_j من الجملة (الكلمة التي يُوْشر إليها مؤشر بداية الجملة):
 - إذا كان محتوى الخانة $Action[s_i, a_j] = S_k$ أي عملية سحب مرفقة بالرقم k يكون:
 - ادفع الرمز a_j إلى أعلى المكس وادفع وراءه بالحالة s_i .
 - تقدم كلمة واحدة إلى الأمام في الجملة.
 - إذا كان محتوى الخانة $Action[s_i, a_j] = R_k$ أي عملية اختصار مرفقة بالرقم k يكون:
 - اسحب من أعلى المكس خانة عددها: $2 * (\text{طول السلسلة اليمينية من القاعدة رقم } k)$.
 - ادفع إلى أعلى المكس بمحتوى $GOTO[s_m, a_n]$ حيث a_n هو الرمز اليساري للقاعدة رقم k، و s_m هو الرمز الموجود في أعلى المكس بعد تنفيذ عملية التفريغ السابقة.
- نكرر العمل طالما لم نصل إلى نهاية الجملة.

b. بناء جدول التحليل

سنبنى جدول التحليل أيضاً اعتماداً على المثال الذي استخدمناه في الفقرة السابقة. ليكن لدينا النحو الصرفي:

$$\begin{cases} E \rightarrow E + T \\ E \rightarrow T \\ T \rightarrow T * F \\ T \rightarrow F \\ F \rightarrow (E) \\ F \rightarrow Id \end{cases}$$

سنفترض الآن أننا سنتعامل مع القواعد بصيغة الحالات وأن لدينا حالة ابتدائية تحتوي على جميع القواعد مع مؤشر لقراءة الرموز نرسم له بنقطة (•) فيكون للقواعد الشكل التالي وندعوها الحالة I_0 :

$$I_0 = \begin{cases} E \rightarrow \bullet E + T \\ E \rightarrow \bullet T \\ T \rightarrow \bullet T * F \\ T \rightarrow \bullet F \\ F \rightarrow \bullet (E) \\ F \rightarrow \bullet Id \end{cases}$$

أولاً- الإغلاق على مجموعة من القواعد التي تشكل حالة:

إذا كان لدينا مجموعة من القواعد التي تشكل حالة ما I ، فيمكن حساب الإغلاق على هذه الحالة، والذي ندعوه $Closure(I)$ وفقاً للخوارزمية التالية:

$$\begin{aligned} & \diamond \text{ نضع كل عنصر من الحالة } I \text{ ضمن } Closure(I). \\ & \diamond \text{ من أجل كل عنصر من } Closure(I) \text{ له الشكل: } A \rightarrow \alpha \bullet B\beta \\ & \quad \circ \text{ من أجل كل قاعدة } B \rightarrow \gamma \text{ } \\ & \quad \blacksquare \text{ نضع هذه القاعدة في } Closure(I). \\ & \diamond \text{ كرر العمل حتى يتم إغلاق } Closure(I) \text{ ولا يعود هناك ما نضيفه.} \end{aligned}$$

$$I = \begin{cases} E \rightarrow E \bullet + T \\ T \rightarrow T * \bullet F \end{cases} \text{ فمثلاً، اعتباراً من النحو السابق ومن}$$

$$Closure(I) = \begin{cases} E \rightarrow E \bullet + T \\ T \rightarrow T * \bullet F \\ F \rightarrow \bullet (E) \\ F \rightarrow \bullet Id \end{cases} \text{ يكون لدينا}$$

ثانياً- تعريف الانتقال من حالة إلى حالة:

نعرف الانتقال من حالة I_j إلى حالة أخرى I_k وذلك بعد قراءة رمز X كما يلي:

$$I_k = \Delta(I_j, X) = Closure(\{A \rightarrow \alpha X \bullet \beta \mid [A \rightarrow \alpha \bullet X\beta] \in I_j\})$$

فمثلاً، تكون نتيجة الانتقال من $I_j = \begin{cases} E \rightarrow E \bullet + T \\ T \rightarrow T \bullet * F \\ F \rightarrow \bullet (E) \\ F \rightarrow \bullet Id \end{cases}$ بقراءة الرمز F هي: $I_k = \{T \rightarrow T * F \bullet\}$

في حين تكون نتيجة الانتقال من $I_j = \begin{cases} E \rightarrow E \bullet + T \\ T \rightarrow T \bullet * F \\ F \rightarrow \bullet (E) \\ F \rightarrow \bullet Id \end{cases}$ بقراءة الرمز + هي: $I_m = \begin{cases} E \rightarrow E + \bullet T \\ T \rightarrow \bullet T * F \\ T \rightarrow \bullet F \\ F \rightarrow \bullet (E) \\ F \rightarrow \bullet Id \end{cases}$

ثالثاً- بناء حالات التحليل (الحالات التي وضعنا أرقامها في العمود اليساري من الجدول):

- ❖ أضف القاعدة $S' \rightarrow S$ (حيث S هو رمز البداية).
- ❖ أحسب $I_0 = \text{Closure}(S' \rightarrow \bullet S)$
- ❖ أضف I_0 إلى مجموعة الحالات.
- ❖ من أجل كل حالة I من مجموعة الحالات:
 - من أجل كل رمز X من رموز النحو حيث $\Delta(I, X) \neq \phi$:
 - أضف $\Delta(I, X)$ إلى مجموعة الحالات.

رابعاً- بناء الجدول:

- ❖ ترقيم القواعد الصرفية.
- ❖ بناء مجموعة الحالات $\{I_0, \dots, I_n\}$.
- ❖ من أجل كل حالة $I_j = \Delta(I_i, a)$ حيث a رمز أولي:
 - $Action[i, a] = Sj$
- من أجل كل حالة $I_j = \Delta(I_i, A)$ حيث A رمز وسيط:
 - $GoTq[i, A] = j$
- من أجل كل حالة $A \rightarrow \alpha \bullet$ محتواة في I_i :
 - من أجل كل $a \in \text{Follow}(A)$
- $Action[i, a] = Rk$ حيث k هي رقم القاعدة ضمن لائحة القواعد الصرفية.

٥. الأخطاء الصرفية (Syntax Errors)

- هناك العديد من الأخطاء التي قد تأتي من عدم كتابة المفردات بالشكل الصحيح (أخطاء تحليل مفرداتي)، أو من عدم تركيب الجمل بالشكل الصحيح (أخطاء صرفية). بجميع الأحوال، يجب على معالج الأخطاء أن:
- يحدد وجود الخطأ بشكل واضح ودقيق.
 - يعالج الخطأ بسرعة حرصاً على سرعة استكمال التحليل.
 - يعالج الخطأ بطريقة فعالة دون أن يؤدي الأمر لتوليد خطأ جديد.

لحسن الحظ، من السهل معالجة الأخطاء المعتادة الناجمة عن فقدان حرف أو كلمة أو استبدال كلمة بأخرى (مثل فقدان فاصلة منقوطة، استخدام فاصلة عادية عوضاً عن فاصلة منقوطة أو بالعكس، ...)، وهي أخطاء لا تحتاج لآليات معقدة لمعالجتها. بكل الأحوال، يمكن لمثل هذه الأخطاء أن تكون موجودة قبل فترة طويلة من اكتشافها، فعلى سبيل المثال لا يظهر فقدان القوس المخصص لبداية أو نهاية مقطع إلا عند نهاية المقطع، مما يجعل من الصعب اكتشاف مصدر الخطأ فيصبح برنامج التحليل مضطرباً "لتوقع" ما يوجد في رأس المبرمج الذي كتب البرنامج. على كل حال توجد استراتيجيات متعددة لمعالجة الأخطاء، ولكن من المهم الانتباه إلى ضرورة إيقاف التحليل عند تجميع عدد كبير من الأخطاء. إذ يصبح عندها من غير المفيد الاستمرار كون الأخطاء التي سيتم اكتشافها لاحقاً ستكون نتيجة للأخطاء السابقة. وعليه هناك عدة استراتيجيات لمتابعة ومعالجة الأخطاء، وهي الحالات التي نستعرضها فيما يلي:

a. أسلوب Panic Mode

وهو الأبسط في المعالجة، إذ يقوم المحلل، عند مواجهته لخطأ ما، بمتابعة القراءة من خلال حذف رموز الدخل الواحد تلو الآخر حتى يصل إلى إحدى المفردات الأساسية التي تشكل نهاية جملة أو مقطع (مثل ";" أو "end" أو "{ ...") والتي يكون لها دور واضح في البرنامج المصدري الذي يجري تحليله. ومن هناك يبدأ من جديد مع تحديد وجود خطأ في بداية المقطع السابق. يتميز هذا الأسلوب ببساطته، ولكن نتائج الأخطاء التي يعطيها تكون غير مفصلة وغير دقيقة وتترك قسماً كبيراً من النص المصدري دون تحقق.

b. تصحيح الأخطاء

يمكن في بعض الحالات محاولة تصحيح بعض الأخطاء. فعند تنفيذ التحليل القواعدي باستخدام خوارزميات التحليل التي سبق وذكرناها، يمكننا واعتماداً على الجداول، محاولة أتمتة عملية توقع خطأ. ففي جميع الحالات التي درسناها، يمتلك جدول القواعد (أو جدول التحليل) خانة فارغة، ويكون وقوع خوارزمية التحليل على إحداها معبراً عن اكتشاف خطأ ما. مما يسمح لنا بتوقع الحالات التي يمكن أن تقع فيها خوارزمية التحليل على خانة فارغة وتوقع تصحيحها.

إلا أن سيئة هذه العملية تكمن في أن العديد من الأخطاء التي قد تسبب وقوع خوارزمية التحليل في خانة فارغة، قد تكون ناشئة أصلاً عن خطأ يسبق بكثير الوضع الحالي، مما يعني أن أي تصحيح قد يكون بدوره خاطئاً.

c. إضافة قواعد للأخطاء

يمكننا معالجة العديد من الأخطاء الشائعة في حال كان لدينا فكرة واضحة عنها، عبر إضافة قواعد تحدد الخطأ. فعلى سبيل المثال عند وجود جملة شرطية مثل `if (Expression) then INST else INST` يمكننا إضافة قاعدة خطأ مثل: `I → if Expression then (Error – No Parenthesis)`.

٦. مسألة

باستخدام النحو الصرفي G سنقوم بتوصيف مجموعة جزئية من اللغة الإنكليزية.

- سنفرض أن لدينا مجموعة الرموز المنتهية (Terminals) التالية: $T = \{ \text{Verb, Name, who, and} \}$

- يأخذ Verb إحدى القيم التالية:
- Verb = { loves, follows, slaps, listen to, deranges }
- يأخذ Name إحدى القيم التالية:
- Name={ Marc, Brad, Bob, Anne, Sophie }
- لا تأخذ الرموز { who, and } إلا قيمة واحدة هي قيمتها نفسها.
- لنفرض أن الرموز الوسيطة (Non Terminals) في هذا النحو هي: $N=\{P, S, C, R\}$
- يمثل الرمز P الجملة (Sentence)؛
- يمثل الرمز S الفاعل (Subject)؛
- يمثل الرمز C المفعول به (Complement)؛
- يمثل الرمز R حروف الوصل (Subordinates).
- تكون القواعد الصرفية الخاصة بهذا النحو على الشكل التالي:

- (1) $P \rightarrow S \text{ verb } C$
- (2) $S \rightarrow \text{Name}$
- (3) $C \rightarrow \text{Name}$
- (4) $C \rightarrow \text{Name } R$
- (5) $R \rightarrow \text{who verb } C$
- (6) $R \rightarrow R \text{ and } R$

الأسئلة:

1. نريد تحليل الجملة التالية (بغض النظر عن تصريف الأفعال):
« *Marc follows Bob who listen to Anne who deranges Brad and who loves Sophie* »
 - a. أعط شجرة اشتقاق خاصة بهذه الجملة.
 - b. برأيك هل النحو المقترح غامض؟ علل إجابتك.
2. عدل القواعد لإلغاء العودية اليسارية وإلغاء المعاملات المشتركة.
3. لندعو $G1$ النحو الصرفي الناتج بعد إلغاء العودية اليسارية والمعاملات المشتركة.
 - a. قم ببناء منظومة التحليل Top-Down Parsing من النمط $LL(1)$ للنحو $G1$ من خلال بناء جدول القواعد (أو جدول المناقلات Production Table).
4. اعتباراً من النحو G سنقوم بإلغاء القاعدة الأخيرة للرمز R (القاعدة رقم 6) لنحصل على النحو $G2$:
 - a. قم ببناء منظومة تحليل Bottom-Up Parsing للنحو $G2$ من خلال بناء جداول ACTION و GOTO.

الحل:

الجواب الأول:

« *Marc follows Bob who listen to Anne who deranges Brad and who loves Sophie* »

a. نطبق عملية اشتقاق يساري للقواعد:

P →

S Verb C →

Marc Verb C →

Marc follows C →

Marc follows Name R →

Marc follows Bob R →

Marc follows Bob who Verb C →

Marc follows Bob who listen to C →

Marc follows Bob who listen to Name R →

Marc follows Bob who listen to Anne R →

Marc follows Bob who listen to Anne R and R →

Marc follows Bob who listen to Anne who verb C and R →

Marc follows Bob who listen to Anne who deranges C and R →

Marc follows Bob who listen to Anne who deranges Name and R →

Marc follows Bob who listen to Anne who deranges Brad and R →

Marc follows Bob who listen to Anne who deranges Brad and who Verb C →

Marc follows Bob who listen to Anne who deranges Brad and who loves C →

Marc follows Bob who listen to Anne who deranges Brad and who loves Sophie

b. نلاحظ أن القواعد غامضة والبرهان وجود مسار اشتقاق يساري آخر لنفس الجملة:

P →

S Verb C →

Marc Verb C →

Marc follows C →

Marc follows Name R →

Marc follows Bob R and R → هنا يبدأ المسار المختلف

Marc follows Bob who Verb C and R →

Marc follows Bob who listen to C and R →

Marc follows Bob who listen to Name and R →

Marc follows Bob who listen to Anne R and R →

Marc follows Bob who listen to Anne R and R →

Marc follows Bob who listen to Anne who verb C and R →

Marc follows Bob who listen to Anne who deranges C and R →

Marc follows Bob who listen to Anne who deranges Name and R →

Marc follows Bob who listen to Anne who deranges Brad and R →

Marc follows Bob who listen to Anne who deranges Brad and who Verb C →

Marc follows Bob who listen to Anne who deranges Brad and who loves C →

Marc follows Bob who listen to Anne who deranges Brad and who loves Sophie

الجواب الثاني:

نطبق خوارزمية إلغاء العودية اليسارية المباشرة على القاعدة الوحيدة التي تحتويها:

$$\begin{array}{l} P \rightarrow S \text{ Verb } C \\ S \rightarrow \text{Name} \\ C \rightarrow \text{Name} \mid \text{Name } R \\ R \rightarrow R \text{ and } R \mid \text{who Verb } C \end{array} \rightarrow \begin{array}{l} P \rightarrow S \text{ Verb } C \\ S \rightarrow \text{Name} \\ C \rightarrow \text{Name} \mid \text{Name } R \\ R \rightarrow \text{who Verb } C R' \\ R' \rightarrow \text{and } R R' \mid \varepsilon \end{array}$$

نطبق خوارزمية إلغاء العودية اليسارية (غير المباشرة):

$$\begin{array}{l} P \rightarrow S \text{ Verb } C \\ S \rightarrow \text{Name} \\ C \rightarrow \text{Name} \mid \text{Name } R \\ R \rightarrow \text{who Verb } C R' \\ R' \rightarrow \text{and } R R' \mid \varepsilon \end{array} \rightarrow \begin{array}{l} P \rightarrow \text{Name Verb } C \\ C \rightarrow \text{Name} \mid \text{Name } R \\ R \rightarrow \text{who Verb } C R' \\ R' \rightarrow \text{and } R R' \mid \varepsilon \end{array}$$

نطبق خوارزمية حساب المعاملات المشتركة (في حال وجدت) في نحو الصرفي:

$$\begin{array}{l} P \rightarrow \text{Name Verb } C \\ C \rightarrow \text{Name} \mid \text{Name } R \\ R \rightarrow \text{who Verb } C R' \\ R' \rightarrow \text{and } R R' \mid \varepsilon \end{array} \rightarrow \begin{array}{l} P \rightarrow \text{Name Verb } C \\ C \rightarrow \text{Name } C' \\ C' \rightarrow R \mid \varepsilon \\ R \rightarrow \text{who Verb } C R' \\ R' \rightarrow \text{and } R R' \mid \varepsilon \end{array}$$

الجواب الثالث:

نقوم بحساب $\text{First}(X)$ من أجل كل رمز وسيط X ينتمي إلى النحو الصرفي:

$$\begin{array}{l} P \rightarrow \text{Name Verb } C \\ C \rightarrow \text{Name } C' \\ C' \rightarrow R \mid \varepsilon \\ R \rightarrow \text{who Verb } C R' \\ R' \rightarrow \text{and } R R' \mid \varepsilon \end{array} \rightarrow \begin{array}{l} \text{First}(P) = \text{First}(C) = \{\text{Name}\} \\ \text{First}(C') = \{\text{who}, \varepsilon\} \\ \text{First}(R) = \{\text{who}\} \\ \text{First}(R') = \{\text{and}, \varepsilon\} \end{array}$$

نقوم بحساب $\text{Follow}(X)$ من أجل كل رمز وسيط X ينتمي إلى النحو الصرفي:

$$\begin{array}{l} P \rightarrow \text{Name Verb } C \\ C \rightarrow \text{Name } C' \\ C' \rightarrow R \mid \varepsilon \\ R \rightarrow \text{who Verb } C R' \\ R' \rightarrow \text{and } R R' \mid \varepsilon \end{array} \rightarrow \begin{array}{l} \text{Follow}(P) = \text{Follow}(R) = \text{Follow}(R') = \{\$\} \\ \text{Follow}(C) = \text{Follow}(C') = \{\$, \text{and}\} \end{array}$$

نقوم ببناء الجدول:

	Name	Verb	who	and	\$
P	$P \rightarrow \text{Name Verb } C$				
C	$C \rightarrow \text{Name } C'$				
C'			$C' \rightarrow R$	$C' \rightarrow \varepsilon$	$C' \rightarrow \varepsilon$
R			$R \rightarrow \text{who Verb } C R'$		
R'				$R' \rightarrow \text{and } R R'$	$R' \rightarrow \varepsilon$

الجواب الرابع:

يكون النحو G2 على النحو التالي:

$$\begin{array}{l} P \rightarrow S \text{ Verb } C \\ S \rightarrow \text{Name} \\ C \rightarrow \text{Name} \mid \text{Name } R \\ R \rightarrow \text{who } \text{Verb } C \end{array}$$

لبناء الجدول، نبدأ بتقييم القواعد الصرفية وفقاً للخوارزمية المعتمدة لبناء جدول التحليل:

$$\begin{array}{l} (1) P \rightarrow S \text{ Verb } C \\ (2) S \rightarrow \text{Name} \\ (3) C \rightarrow \text{Name} \\ (4) C \rightarrow \text{Name } R \\ (5) R \rightarrow \text{who } \text{Verb } C \end{array}$$

لنبدأ ببناء الحالات وفقاً لخوارزمية توليد الحالات حيث نعتبر أن قاعدة P هي القاعدة الابتدائية كونها وحيدة ونحسب الإغلاق على P:

$$I_0 = \left\{ \begin{array}{l} P \rightarrow \bullet S \text{ Verb } C \\ S \rightarrow \bullet \text{Name} \end{array} \right.$$

$$I_0 = \left\{ \begin{array}{l} P \rightarrow \bullet S \text{ Verb } C \\ S \rightarrow \bullet \text{Name} \end{array} \right. \rightarrow \begin{array}{l} I_1 = \Delta(I_0, S) = \{P \rightarrow S \bullet \text{Verb } C \\ I_2 = \Delta(I_0, \text{Name}) = \{S \rightarrow \text{Name} \bullet \end{array}$$

$$I_1 = \{P \rightarrow S \bullet \text{Verb } C\} \rightarrow I_3 = \Delta(I_1, \text{Verb}) = \left\{ \begin{array}{l} P \rightarrow S \text{ Verb} \bullet C \\ C \rightarrow \bullet \text{Name } R \\ C \rightarrow \bullet \text{Name} \end{array} \right.$$

$$I_3 = \left\{ \begin{array}{l} P \rightarrow S \text{ Verb} \bullet C \\ C \rightarrow \bullet \text{Name } R \\ C \rightarrow \bullet \text{Name} \end{array} \right. \rightarrow \begin{array}{l} I_4 = \Delta(I_3, C) = \{P \rightarrow S \text{ Verb } C \bullet \\ I_5 = \Delta(I_3, \text{Name}) = \left\{ \begin{array}{l} C \rightarrow \text{Name} \bullet \\ C \rightarrow \text{Name} \bullet R \\ R \rightarrow \bullet \text{who } \text{Verb } C \end{array} \right. \end{array}$$

$$I_5 = \left\{ \begin{array}{l} C \rightarrow \text{Name} \bullet \\ C \rightarrow \text{Name} \bullet R \\ R \rightarrow \bullet \text{who } \text{Verb } C \end{array} \right. \rightarrow \begin{array}{l} I_6 = \Delta(I_5, R) = \{C \rightarrow \text{Name } R \bullet \\ I_7 = \Delta(I_5, \text{who}) = \{C \rightarrow \text{who} \bullet \text{Verb } C \end{array}$$

$$I_7 = \{C \rightarrow \text{who} \bullet \text{Verb } C\} \rightarrow \Delta(I_7, \text{Verb}) = I_3$$

لنحسب المجموعات Follow(X) حيث X هو رمز وسيط من النحو الصرفي:

- (1) P → S Verb C
- (2) S → Name
- (3) C → Name
- (4) C → Name R
- (5) R → who Verb C



- Follow(P) = {\$}
- Follow(S) = {Verb}
- Follow(C) = {\$}
- Follow(R) = {who,\$}

ويكون الجدول بعد تطبيق خوارزمية بنائه كما يلي:

	ACTION				GOTO			
	Name	Verb	who	\$	P	S	C	R
0	S2					1		
1		S3						
2		R2						
3	S5						4	
4			R1	R1				
5			S7	R3				6
6			R4	R4				
7		S3						

الفصل السابع

الأداة Yacc/Bison (أنظر الأدوات المرافقة)

نظراً لأن عملية التحليل الصرفي لجمال البرنامج المصدري، قابلة للبرمجة بعد أن تم بناء كافة الخوارزميات اللازمة لتنفيذ تحليل صرفي نازل أو صاعد اعتماداً على جداول خاصة وعلى بنى أوتومات ذات مكس، قام مصمموا المترجمات ببناء العديد من الأدوات التي تسمح ببناء المحلل القواعدي الصرفي آلياً اعتباراً من توصيف للنحو الصرفي الذي يحدد قواعد اللغة.

من أشهر الأدوات التي تم تصميمها، كانت أداة YACC تحت أنظمة UNIX والتي تم اشتقاق الأداة Bison منها لتعمل في بيئة العمل Window. تقوم هذه الأداة على فكرة توصيف القواعد الصرفية للغة البرمجة المصدريّة باستخدام لغة توصيف خاصة ندعوها (Backus Naur Form) ضمن ملف خاص، ومن ثم تشغيل الأداة التي تقرأ هذا الملف لتولد برنامجاً بلغة البرمجة C يمثل المحلل الصرفي باستخدام التحليل الصاعد من النمط LALR وهو تحليل مشتق من التحليل LR مع قراء رمز واحد إضافي عند تنفيذ عملية التحليل لذلك يدعى هذا النوع من التحليل (LALR: Look Ahead Symbol Left Scanning Rightmost Derivation).

1 . بنية ملف توصيف القواعد الصرفية

تكون بنية ملف Bison على الشكل التالي:

```
%  
{  
Declaration (in C language) of variables, constants, included files, ...  
%}  
Declaration of used lexical units  
Declaration of priorities  
Declaration of types  
%%  
Production Rules & Semantic Actions  
%%  
Useful routines and functions written in C language and main function
```

- يتضمن الملف مجموعة الرموز الأولية (Terminals) الخاصة بالنحو والتي يمكن أن نعرفها بإحدى الأشكال التالية:
 - كمفردات (Lexical Units) والتي يتم تعريفها مسبوقة بكلمة %token مثل: %token else
 - كمحارف، ونضعها بين فواصل مثل: 'a', ..., '+'
 - سلاسل الأحرف والتي نضعها بين فواصل مضاعفة مثل: "while", ..., "+=".
- يتضمن الملف مجموعة الرموز الوسيطة (Non Terminals) والتي تظهر عند تعريف القواعد الصرفية للدلالة على نوع التعليمات أو العملية التي تمثلها القاعدة، وتكون عبارة عن سلاسل من المحارف غير معرفة كمفردات أو كرموز نهائية في اللغة. من الجدير بالذكر أن Bison يفرق بين الأحرف الكبيرة والصغيرة لذا

يمكن استخدام السلسلة IF مثلاً للدلالة على قاعدة الشرط بحيث لا تكون مكافئة للسلسلة "if" التي تدل على الكلمة المفتاحية والتي نتعامل معها كرمز أولي.

- مجموعة القواعد الصرفية، وهي عبارة عن سلسلة من التعليمات من الشكل:

```
Non_Terminal : Prod1
                | Prod2
                ...
                | Prodn
```

- مجموعة العمليات المرافقة للقواعد والتي يتم تنفيذها عند تنفيذ التحليل (وتتعلق بعملية التحليل الدلالي وبناء بنى المعطيات المساعدة):

```
P : A {printf("Reduce by A");}
| S B 'X' {printf("Known Character X");}
```

- القسم الخاص بالإجرائيات والتتابع التي يمكن أن تساعد في التحليل الصرفي.

٢ . التوصل مع محلل المفردات: yyval

يتواصل المحلل القواعدي الصرفي مع المحلل المفرداتي عبر المتحول المشترك yyval والذي يكون من النمط YYSTYPE وهو نمط عدد صحيح. فضمن ملف دخل الأداة (flex) ترسل تعليمة return(unit) للمحلل القواعدي مفردة هي unit. تكون قيمة هذه المفردة مخزنة في المتحول المشترك yyval. بالتالي، يأخذ المحلل القواعدي محتوى yyval على أنه المفردة الجديدة (يتعامل معها كرقم في الحالة التلقائية). يمكن تغيير نمط yyval مثلاً إلى نمط آخر عبر تعريف:

```
#define YYSTYPE new_Type_in_C
```

أو بتعريف النمط YYSTYPE باستخدام التعليمة union % التي تعدله إلى:

```
%union {
    int Integer;
    double Real;
    char* String;
}
```

عندها يصبح yyval قادر على تخزين قيم منمطة سواء كانت أعداد صحيحة أو حقيقية أو سلاسل محارف. ويمكننا اعتماداً على ما سبق تحديد أنماط المفردات التي يعيدها المحلل المفرداتي من خلال محتوى البنية السابقة، فنضع:

```
%token <Integer> NUMBER
%token <String> IDENT
```

٣ . المتحولات، والإجرائيات، والتتابع المعرفة

Variable/Function	توصيف الاستخدام
YYACCEPT	تعلية تسمح بإيقاف المحلل القواعدي. تعيد عندها yyparse القيمة 0 والتي تعني نجاح التحليل

YYABORT	تعليلة تسمح بإيقاف المحلل القواعدي. تعيد عندها yyparse القيمة 1 والتي تعني فشل التحليل
Yyparse()	استدعاء المحلل القواعدي
main()	وهي الإجرائية الرئيسية التي يمكن أن تقتصر على استدعاء () yyparse، ويمكن أن يضيف المبرمج إليها استدعاءات أخرى.
%start non-terminal	للإشارة إلى القاعدة الأولى في سلسلة القواعد الصرفية

٤. حالات التَضارب *Shift/Reduce* والتضارب *Reduce/Reduce*

عند تنفيذ عملية التحليل الصاعد، يمكن لجدول الـ ACTION أن يحتوي في إحدى خاناته على حالات تضارب *Shift/Reduce* أو *Reduce/Reduce*. في حالة التضارب الأول من نمط *Shift/Reduce*، يكون المحلل القواعدي أمام خيارين هما: تنفيذ عملية *Shift* (سحب) أو تنفيذ عملية *Reduce* (اختصار). أما في حالة التضارب الثاني من نمط *Reduce/Reduce*، فيكون المحلل القواعدي أمام خيارين هما القيام بإحدى عمليتي *Reduce* (اختصار) مختلفتين.

```
> bison example.y
conflicts: 6 shift/reduce, 2 reduce/reduce
```

يتعامل Bison مع هذه المشكلة (في حال لم يعدل المصمم من قواعده) بالشكل التالي:

- في حالة التضارب *Reduce/Reduce* يختار المحلل القاعدة التي تظهر أولاً في توصيف سلسلة القواعد لينفذ باستخدامها عملية الاختصار.
- في حالة التضارب *Shift/Reduce* يقوم المحلل بتفضيل عملية السحب *Shift*. يمكن الرجوع إلى الجدول الذي يولده Bison في الملف `y.table` عند تنفيذ التحليل باستخدام الخيار `-v` لمعرفة كيفية حل التضاربات.

٥. تحديد أفضليات القواعد وطرق تجميعها

يمكن حل العديد من مشاكل التضارب وخصوصاً تلك التي تظهر في القواعد الصرفية الخاصة بتوصيف العمليات الحسابية، وذلك عن طريق تحديد طرق تجميع العمليات وأفضليتها بالنسبة لبعضها البعض. فعلى سبيل المثال:

```
%left term1 term2
%right term3
%left term4
%nonassoc term5
```

تشير إلى أن `term1` و `term2` و `term4` تجميعية يسارية، في حين أن `term3` تجميعية يمينية، وأن `term5` ليس تجميعياً. كما يمكن إعطاء أفضليات بعض القواعد عبر تعريفها بالشكل:

```
%prec terminal-or-lexical-unit
```

- تساعد الأفضليات وتحديد النمط التجميعي للقواعد في حل بعض مشاكل التضاربات، فعلى سبيل المثال، في حال وجود تضارب بين اختصار بالقاعدة $A \rightarrow \alpha$ والسحب باستخدام رمز b ، يمكن حله بتطبيق مايلي:
- إذا كانت أفضلية القاعدة A أعلى من أفضلية الرمز b ، يطبق المحلل الاختصار.
 - إذا كان لهما نفس الأفضلية وكانت القاعدة تجميعية إلى اليسار، يطبق المحلل الاختصار.
 - في بقية الحالات، يطبق المحلل السحب.

٦. نموذج عن ملف *.Y *

فيما يلي ملف دخل لأداة Bison يحتوي على القواعد الصرفية للغة مؤلفة من جمل يكون فيها إما عدد زوجي من الكلمة a أو عدد فردي من الكلمة b أو كليهما معاً:

```
%%
WORD : PI '$' {printf("accepted word\n");YYACCEPT;}
      ;

PP : 'a' IP
   | 'b' PI
   | /* empty */
   ;
IP : 'a' PP
   | 'b' II
   | 'a'
   ;
PI : 'a' II
   | 'b' PP
   | 'b'
   ;
II : 'a' PI
   | 'b' IP
   | 'a' 'b'
   | 'b' 'a'
   ;
%%
int yylex() {
    char car=getchar();
    if (car=='a' || car=='b' || car=='$') return(car);
    else printf("ERROR : Non recognized Character : %c ",car);
}
```

الفصل الثامن

التحليل الدلالي

هناك مجموعة من خصائص لغات البرمجة التي لا يمكن توصيفها باستخدام النحو خارج السياق الذي استخدمناه لتوصيف القواعد الصرفية للغة، لأنها وببساطة خصائص ترتبط ارتباط مباشر بسياق البرنامج المكتوب بهذه اللغة. فعلى سبيل المثال لا الحصر: لا يمكننا الإعلان عن متحول مرتين في نفس الإجراءية أو المقطع البرمجي (سنعرف مفهوم المقطع لاحقاً)، كما لا يمكننا استخدام متحول دون الإعلان عنه بشكل مسبق (في بعض لغات البرمجة). كما لا يمكننا أن نبرمج عملية جداء عدد حقيقي بسلسلة محارف. يهتم المحلل الدلالي (أو محلل السياق) بالتحقق من الخصائص المرتبطة بالسياق للغة البرمجة، ويجري تنفيذ عملية التحليل الدلالي في نفس الوقت الذي تتم فيه عملية التحليل القواعدي الصرفي وذلك اعتماداً على إجراءات وعمليات يتم استدعاؤها ضمن القواعد الصرفية.

عموماً، لا توجد طريقة وأسلوب وبنى محددة لتنفيذ التحليل الدلالي. فالعملية ترتبط من ناحية، بلغة البرمجة وقواعدها، وترتبط من ناحية أخرى بأسلوب مصمم المترجم وطريقة تصميمه لبنى المعطيات التي ستستقبل معلومات السياق (مثل المتحولات وأنماطها عند الإعلان عنها)، والإجراءات التي سيتم استدعاؤها (إجراءية التحقق من صلاحية عملية جداء مثلاً بين تعبيرين رياضيين).

١ . مجال تعريف ورؤية المتحولات

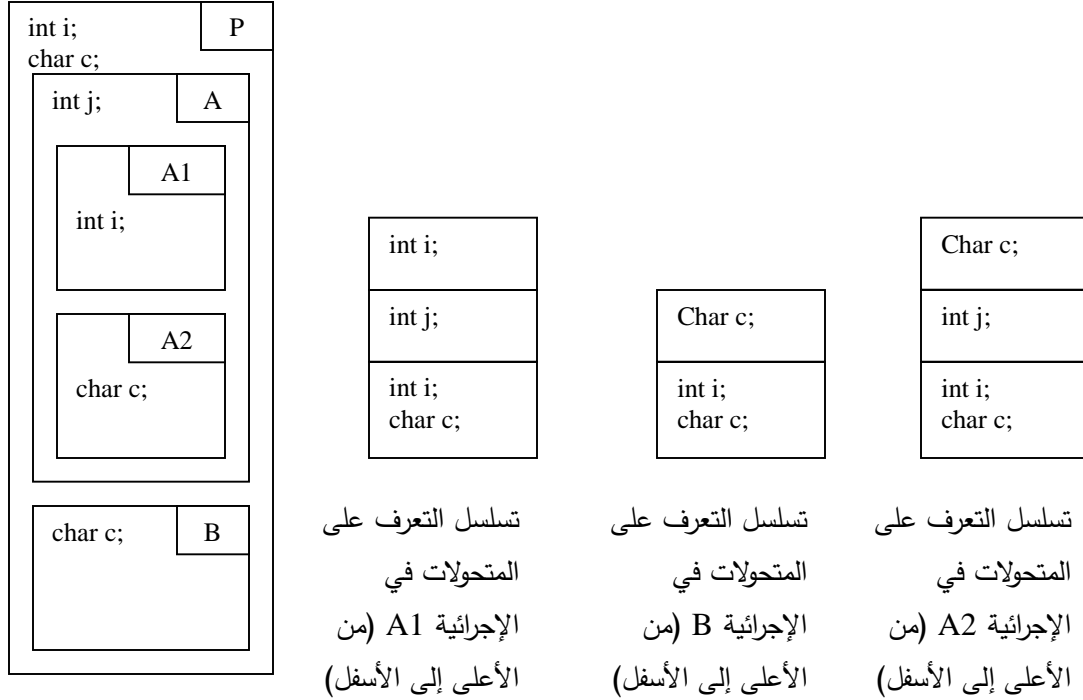
نعرف مجال تعريف المتحول بأنه مجموعة أجزاء البرنامج الذي يكون المتحولاً فيها معروفاً وقابلاً للاستخدام وفقاً للمعنى الذي أعطي له عند الإعلان عنه. يختلف معنى هذا المجال من لغة إلى أخرى، ففي لغة كوبول (COBOL) تكون التحولات معروفة ومرئية في كافة أجزاء البرنامج، في حين لا تكون هذه المتحولات معروفة أو مرئية إلا في المقاطع التي يجري فيها الإعلان عن المتحول في لغات برمجة مثل باسكال (Pascal) و لغة سي (C).

للمساعدة في تحديد مجال تعريف ورؤية المتحول، يستخدم مصمم المترجم بنية معطيات رئيسية هي جدول الرموز (Symbol Table)، يكون هدفها تخزين المتحول عند الإعلان عنه مع تحديد بعض العناصر المرتبطة به وخصوصاً نمطه والمقطع الذي ينتمي إليه. لذا يحتاج مصمم هذه البنية إلى وضع بنية تسمح له -عند قيامه بالتحليل الدلالي أثناء التحليل القواعدي، وعند تحليله لتعبير رياضي يستخدم متحول ما- أن يعود إلى جدول الرموز للتأكد من أن هذا المتحول مرئي ومعرف في المكان الذي تم استخدامه فيه.

تحتوي بنية رمز في جدول الرموز، على المعلومات التالية:

- اسم المتحول (X, y, t).
- نمط المتحول (عدد صحيح، عدد حقيقي، ... الخ).
- في حال كان الإعلان عن إجراءية فيتم تحديد عدد معاملاتها وأنماط هذه المعاملات.

يمكن ضمن هذا الجدول الاحتفاظ بمعلومة المقطع الذي ينتمي إليه المتحول، أو بناء الجدول نفسه على شكل مجموعة من الجداول التي يرتبط كل منها بمقطع، وتكون الجداول مرتبطة ببعضها البعض تبعاً لهرمية ارتباط المقاطع ببعضها البعض. في الحالة الأخيرة يتم البحث عن متحول (عند استخدامه) للتأكد من وجود إعلان عنه وللتأكد من نمطه، في جدول رموز المقطع نفسه، أو في جداول رموز المقاطع التي تحتوي هذا المقطع فقط. يوضح الشكل التالي هرمية ارتباط مجموعة من جداول رموز ببعضها البعض ضمن برنامج P يحتوي على إجراءات A وإجرائية B معرفتين فيه، وتم تعريف إجرائيتين إضافيتين A1 و A2 ضمن الإجرائية A.



٢ . التحقق من الأنماط

عندما تكون اللغة المصدرية منمطة، يجب أن يتحقق المترجم من صحة العمليات التي تتم على هذه الأنماط من حيث ملاءمتها للنمط المعرف. يختلف هذا التحقق من لغة برمجة إلى أخرى وتكون قواعد الترميز والقواعد الدلالية الأخرى من مسؤولية مصمم اللغة. فعلى سبيل المثال، لا يمكن في لغة مثل لغة البرمجة C جمع عدد حقيقي من النمط double مع سلسلة محارف لها النمط `char*`، كما لا يمكن تنفيذ عملية جداء عدد صحيح `int` ببنية مركبة `struct`. إلا أن بعض العمليات الأخرى في نفس اللغة، تكون ممكنة. فعلى سبيل المثال، يمكن إسناد عدد صحيح `int` إلى متحول من النمط الحقيقي `double` أو من النمط `char`.

عموماً، هناك نوعان من التحقق الدلالي: الأول ساكن يتم في مرحلة الترجمة كالأشكال التي ذكرناها سابقاً والآخر ديناميكي يتم عند التنفيذ ولا يمكن للمترجم مراقبته كأن يكون لدينا جدول معرف بعشر خانات `tab[10]` ونستخدم مؤشر I فيه القيمة ١٢ كمؤشر ضمن هذا الجدول أي أن نكتب `tab[i]=10` مع أن i تحتوي القيمة ١٢.

تتم عملية التحقق من الأنماط عبر حساب أنماط التعابير والمتحولات من خلال إجراءات يتم استدعاءها أثناء عمليات المسح وضمن القواعد الصرفية.

فعل سبيل المثال يمكننا أن نكتب قاعدة صرفية توصف عملية الإسناد ما يلي:

```

I → Id=E
{
  if (Id.Compute_type() == E.Compute_type())
    return true;
  else
    return Error("Incompatible type", LineNb);
}

```

حيث نلاحظ مما سبق أننا نحتاج لمعرفة نمط كل من I و E قبل تنفيذ الإجراءات السابقة. كما يمكننا أن نكتب ضمن قاعدة صرفية خاصة بعملية جمع تعبيرين رياضيين مايلي:

```

E → E + E
{
  if ((E(1).Compute_type() == integer) && (E(2).Compute_type() == integer))
    E(0).SetType(integer)
}

```

يمكننا تعريف قواعد التحقق من الأنماط بلغة محكية أو استخدام توصيف رياضي لها يساعدنا في توضيحها (كما هو الحال في القواعد الصرفية) وبحيث يسهل تحويلها إلى إجراءات وإدراجها ضمن القواعد الصرفية عند الحاجة. لتنفيذ ذلك، لنعرف التدوينين التاليين:

$$\alpha \Rightarrow e : \tau$$

"ضمن السياق a، يمثل e تعبيراً منمطاً، ويكون من النمط τ "

$$\alpha \Rightarrow_L e : \tau$$

"ضمن السياق a، يمثل e (Left-value) تقبل الإسناد، وتكون من النمط τ "

ويمكن اعتباراً من التدوينين السابقين تصميم قواعد التتميط التالية وفق آليات الاستنتاج الموضحة فيما يلي حيث يؤدي تحقق الشروط الموجودة في أعلى كل خط إلى تحقق ما هو وارد في أسفل كل خط. تساعد هذه القواعد عند تصميمها ووضعها بشكل واضح، في تنفيذ إجراءات التحليل الدلالي بشكل أبسط عند برمجة المترجم. فيما يلي بعض الأمثلة عن قواعد حساب أنماط التعابير الرياضية:

Rule	Description
$\frac{\alpha \Rightarrow_L e : \text{int}}{\alpha \Rightarrow ++e : \text{int}}$	ضمن السياق α ، إذا كانت e عبارة عن (Left-value) تقبل الإسناد ومن النمط int، فإن ++e مقبولة ضمن السياق نفسه وتكون من النمط int.
$\frac{\alpha \Rightarrow_L e : \text{int}}{\alpha \Rightarrow ++e : \text{int}}$	ضمن السياق α ، إذا كانت e عبارة عن (Left-value) تقبل الإسناد ومن النمط int، فإن ++e مقبولة ضمن السياق نفسه وتكون من النمط int.

$\begin{array}{l} \alpha \Rightarrow e1 : \tau1 \\ \alpha \Rightarrow e2 : \tau2 \\ \tau1, \tau2 \in \{int, double, float\} \\ op \in \{<, <=, >, >=\} \\ \hline \alpha \Rightarrow e1 \ op \ e2 : boolean \end{array}$	<p>ضمن السياق α، إذا كان $e1$ عبارة عن تعبير من النمط $\tau1$ وكان $e2$ عبارة عن تعبير من النمط $\tau2$ وكانت كلاً من $\tau1$ و $\tau2$ هي إما int أو $double$، أو $float$ فإن تطبيق عملية مقارنة op على التعبيرين، يعطينا تعبيراً من النمط $boolean$ ضمن السياق α.</p>
---	--

الفصل التاسع

توليد الرماز وأمئلته

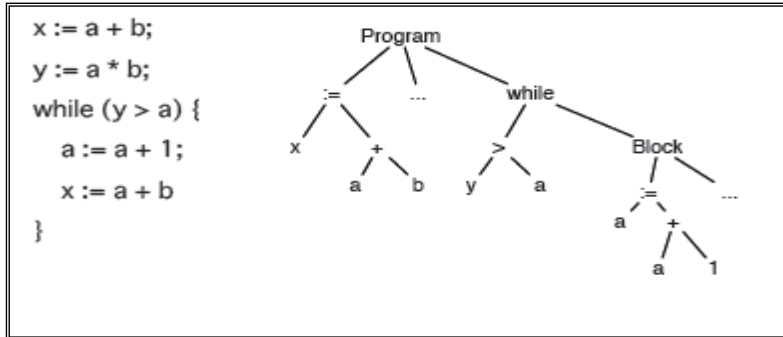
١ . البنية الوسيطة

عند بناء مترجم، يتم استخدام بنية وسيطة لتمثيل البرنامج المصدري الذي تتم ترجمته. تكون هذه البنية الوسيطة على شكل شجرة ندعوها شجرة القواعد المجردة، لأنها تمثل القواعد الصرفية ولكن بصيغة بنية معطيات شجرية. عند انتهاء مسح القواعد الصرفية تكون عملية التحليل المفرداتي والقواعدي والدلالي قد انتهت، وبالإضافة لما سبق، تكون عملية بناء هذه البنية الوسيطة قد انتهت بحيث يمكن اعتباراً منها توليد الرماز عبر عبور الشجرة الممثلة للبرنامج المصدري وتوليد الرماز المناسب.

يمكن لبنية عقدة من الشجرة أن تكون على الشكل التالي:

```
/* types of objects */
typedef enum tempObjType {
    DECL_O, /* declaration */
    STMT_O, /* statement (no return) */
    EXP_O, /* expression (return) */
    LIT_O, /* literal (constant) */
    ID_O /* identifier */
} ObjType;
```

حيث يمثل كل جزء من أجزاء هذه البنية إحدى أشكال العقد التي يمكن أن تكون عليها هذه الشجرة. إذ يمكن للعقد أن تمثل عملية تعريف متحولات، أو جملة تحتوي تعليمة شرطية أو تعليمة حلقية. كما يمكن أن تعبر العقدة عن تعبير رياضي أو منطقي، أو متحولات وعمليات إسناد خاصة بها. فعلى سبيل المثال يمكن لبرنامج أن يمثل بالشكل التالي:



٢ . تنظيم الذاكرة وتنفيذ عملية الحساب

قبل البدء بتوليد الرماز، يجب طرح مجموعة من الأسئلة عن كيفية تنظيم الذاكرة:

- عند إرجاع الإجراءيات لقيم، أو عند استخدام تعليمات القفز من تعليمة مثل تعليمة break في لغة C.
- عند تخزين المتحولات.
- عند الحجز الديناميكي للذاكرة.

هناك بعض الأسئلة التي يتوجب طرحها والتي تكون مرتبطة ارتباط مباشر باللغة المصدريّة التي نترجمها:

- هل يمكن أن تكون الإجراءات، عودية؟
 - هل يمكن لإجرائية أن تستخدم متحولات غير محلية وغير معرفة ضمن مقطعها؟
 - كيف يتم تمرير المعاملات عند استدعاء الإجرائية؟
 - هل يجب أن تكون عملية تحرير مساحة الذاكرة التي يتم حجزها ديناميكياً، عملية يدوية يقوم بها المبرمج الذي يبرمج باللغة المصدرية باستخدام تعليمات خاصة، أم عملية آلية لا تحتاج لتدخل المبرمج؟
 - ما مصير المتحولات المحلية المعرفة ضمن مقطع إجرائية، عند الانتهاء من استدعاء الإجرائية؟
 - كما توجد عدة أسئلة أخرى تتعلق بالنظام واللغة التي تتم عملية الترجمة باتجاهها:
 - ما هو طول عنوان عند حجز الذاكرة؟
 - ما هي الكيانات التي يمكن عنوانها مباشرة في الذاكرة؟
 - ما هي تعليمات الوصول إلى كيانات ومقاطع من الذاكرة قابلة للعنوان المباشرة؟
 - ما هي ضوابط عملية العنوان؟
- تؤثر الإجابات على الأسئلة السابقة في عملية تنظيم الذاكرة. عموماً، يقسم نظام التشغيل الذاكرة المخصصة للبرنامج التنفيذي إلى 4 مناطق أساسية:

الرماز المولد
معطيات ساكنة
مكدس التحكم
منطقة توسع
المكوم

- **منطقة المعطيات الساكنة:** يكون حجم بعض المعطيات معروفاً منذ مرحلة الترجمة، لذ، وعند توليد الرماز، يتم كتابة تعليمات لحجز أماكن لهم وتخزينهم في أماكن محددة سلفاً من الذاكرة وهي الأماكن المخصصة لتخزين المعطيات التي ندعوها ساكنة (Static)، مما يساعد على الانتهاء من توليد تعليمات حجز أماكنهم منذ مرحلة الترجمة وأثناء توليد الرماز.
- **مكدس التحكم:** وهو مكدس يسمح بإدارة عمليات استدعاء الإجراءات وعمليات الإرجاع التي تقوم بها. إذ تمتلك جميع الآلات، آليات خاصة تسمح بكتابة رماز خاص عند استدعاء إجرائية، يقوم هذا الرماز بحجز سجل خاص لحفظ حالة البرنامج الذي قام بالاستدعاء ضمن هذه المنطقة، وإعادته إلى حالته عند انتهاء الاستدعاء.
- **المكوم:** وهو المكان الذي يجري فيه حجز المتحولات المعرفة كمتحولات ديناميكية، بحيث يتم تعريف حجمها أثناء التنفيذ ضمن هذه المنطقة وضمن منطقة التوسع. وتتم إدارة المكوم من قبل الآلة (نظام التشغيل) وفقاً لآليات خاصة تعتمد إما على تنفيذ عمليات تحرير آلي للمساحات المحجوزة عند انتهاء صلاحية المتحولات الخاصة بهذه المساحات، أو على انتظار تعليمات تحرير خاصة يجب أن يتضمنها الرماز المكتوب. كما تقوم الآلة بعملية إدارة المساحة المخصصة للتكويم وفق آليات تنظيم تسمح بأمثلة عملية حجز المساحات فيها والحفاظ دائماً على مساحات كبيرة حرة قابلة للحجز.

بالإضافة لكل ما سبق، نحتاج لإدارة عملية حساب قيم تعابير رياضية ومنطقية، إلى استثمار لغة الخرج لتخزين القيم المرحلية للتعبير قبل الوصول لحساب قيمتها النهائية، مما يعني أننا بحاجة لاستثمار إمكانات التكدس والتخزين والاسترجاع في لغة الخرج بحيث يتم بناء الخرج على نحو يحقق عمليات حساب صحيحة. من أهم آليات الحساب المتبعة والتي تصنف الآلات من خلالها:

- آلية الحساب بالتكدس في الآلة ذات المكس (Stack Machine).

- آلية الحساب بالتخزين في الآلة ذات السجلات (Register Machine).

بشكل عام يمكن إيضاح الفرق بين الآلتين في المثالين التاليين وبحيث سنستعرض بالتفصيل بنية آلة ذات مكس في الفصل الأخير عند الكلام عن الآلة الافتراضية التي سنستخدمها في مشروع المترجم.

يكون رماز خاص بعملية جمع $x+y$ بآلة ذات سجلات على النحو التالي:

STORE Ri x	→ Let the value of register i be stored at address x
LOAD Ri x	→ Fetch the value of x, place it in register i
STORE Rj y	→ Let the value of register j be stored at address y
LOAD Rj y	→ Fetch the value of y, place it in register j
ADD Ri Rj	→ Fetch the value of register j, add it to the value in register i

يكون رماز خاص بعملية جمع $x+y$ بآلة ذات مكس على النحو التالي:

STORE x	→ Store the value of x at address x
STORE y	→ Store the value of y at address y
LOAD x	→ Fetch the value from address x, push it on to the stack
LOAD y	→ Fetch the value from address y, push it on to the stack
ADD	→ Replace the top two values on the stack by their sum

٣. توليد الرماز المقابل للتعليمات

يمكن للجدول التالي أن يوضح لنا، الرماز الواجب توليده بلغة آلة أمام كل نوع من أنواع التعليمات البرمجية الأساسية:

التعليمة البرمجية باللغة المصدرية	التعليمة البرمجية بلغة الآلة
<i>execute(C1;C2)</i>	<i>execute(C1); execute(C2)</i>
<i>execute(if E then C1 else C2)</i>	<i>evaluate E</i> <i>JUMPIF(0) g</i> <i>execute C1</i> <i>JUMP h</i> <i>g: execute C2</i> <i>h:</i>
<i>execute(while E do C)</i>	<i>JUMP h</i> <i>g: execute C</i> <i>h: evaluate E</i> <i>JUMPIF(1) g</i>

	g: evaluate E JUMPFALSE h execute C JUMP g h:
<i>execute(repeat C until E)</i>	g: execute C evaluate E JUMPIF(I) g
<i>execute($I := E$)</i>	evaluate E assign I
<i>execute($L (A)$)</i>	pass-args A CALL p /* $p =$ address of routine $L *I$
<i>pass-args(E)</i> <i>pass-args(@V)</i>	evaluate(E) fetch_Address(V)
<i>fetch(I)</i>	address(I)
<i>assign(I)</i>	STORE address(I)
<i>evaluate($E1$ op $E2$)</i>	evaluate($E1$) evaluate($E2$) op

٤ . الأمثلة

تساعد عملية الأمثلة في الحصول على رماز مختصر مما يؤدي إلى تخفيف استهلاك موارد الآلة التي يتم تنفيذ الرماز عليها سواءً من ناحية تخفيف حجم ذاكرة التخزين المحجوزة أو من ناحية تسريع عملية تنفيذ الرماز. طبعاً ليس من السهل معرفة الأماكن التي يمكن فيها اختصار الرماز، فالعملية تشبه إلى حد ما محاولة توقع ما كان بذهن المبرمج وتحديد أخطاءه. من أهم عمليات الأمثلة التي يمكن أن نقوم بها:

a. تنفيذ مباشر للعمليات على القيم الثابتة اعتباراً من مرحلة الترجمة

والتي تتمثل في تنفيذ الحسابات على القيم الثابتة وعدم بناء شجرة الحساب الخاصة بها دون داع. مثال:

$$A=2+3+A+C; \rightarrow A=5+A+C;$$

$$PI=3.14;$$

$$D=PI/180; \rightarrow D=3.14/180;$$

b. حذف العمليات غير المجدية:

$$x=x+0;$$

$$x=x*1;$$

نشر تعليمة النسخ

يمكن أيضاً اختصار الرمز عبر نشر تعليمة النسخ $x:=y$ (i) وذلك بتعويض كل ظهور للمتحول x بالمتحول y في كل نقطة p من البرنامج إذا فقط إذا:

- كانت التعليمة (i) هي التعريف الوحيد للمتحول x قبل النقطة p .
- من أجل كل مسار من (i) إلى p (بما في ذلك الحلقات) لا يوجد أي تعديل على المتحول y .

c. حساب التعبيرات المشتركة

والتي تظهر في حساب بعض التعبيرات المكررة لمرة واحدة فقط. فعلى سبيل المثال، ليكن لدينا مجموعة التعبيرات:

```
a:=b+c;
b:=a-d;
c:=b+c;
d:=a-d;
```

من الواضح هنا أن التعبيرين الثاني والرابع متشابهان وعليه يمكننا اختصار حساب التعبير الرابع كما يلي:

```
a:=b+c;
b:=a-d;
c:=b+c;
d:=b;
```

هنا يجدر الانتباه إلى أن التعبيرين الأول والثالث لا يحسبان نفس القيمة لأن b تتغير في التعبير الثاني. يمكننا أيضاً ملاحظة المثال التالي:

```
t3:=4*i;
x:=a[t3];
t4:=4*I;
a[t4]:=y;
```

حيث نلاحظ بوضوح أن الحساب الوسيط لكل من $t3$ و $t4$ متشابه مما يسمح لنا بحذف $t4$ والإبقاء على $t3$ وتحويل الرمز إلى الشكل التالي:

```
t3:=4*i;
x:=a[t3];
a[t3]:=y;
```

عموماً، نحتاج لآلية عامة تسمح لنا بتحديد هذه الحالات. يمكننا الاعتماد على البنية الوسيطة المتمثلة بالشجرة لنتمك من فهم هذه الآلية. فمن خلال مسح الشجرة يمكننا تمييز أنواع من التعبيرات:

- **التعبيرات المحسوبة ضمن مقطع:** نقول عن التعبير $(x \text{ op } y)$ أنه محسوب ضمن مقطع B في حال تم تقييمه وحسابه في المقطع B ومن ثم لم يكن هناك أي تعريف لكل من x و y في نفس المقطع. لتنفيذ ذلك نستخدم الخوارزمية التالية:

خوارزمية تحديد التعبيرات المحسوبة ضمن المقطع B:

- ليكن $\text{Prod}(B)$ خالية في البداية:
- من أجل كل تعبير $(x := y \text{ op } z)$ في المقطع B (بالترتيب)
- أضف التعبير $(y \text{ op } z)$ إلى $\text{Prod}(B)$
- احذف أي تعبير يظهر فيه x من $\text{Prod}(B)$.

مثال: من أجل المقطع:

```
a:=b+c;
b:=a-d;
c:=b+c;
d:=a-d;
```

يؤدي تنفيذ الخوارزمية السابقة إلى ما يلي:

Instruction	Bloc
	Empty
a:=b+c	b+c
e:=a+d	b+c, a+d
b:=a-d	a+d, a-d
c:=b+c	a+d, a-d
d:=a-d	Empty

– **التعبيرات المحذوفة ضمن مقطع:** نقول عن التعبير $(x \text{ op } y)$ أنه محذوف من قبل مقطع B في حال تم إعادة تعريف x و y في المقطع B دون إعادة تنفيذ التعبير الحسابي السابق مرة ثانية. لتنفيذ ذلك نستخدم الخوارزمية التالية:

خوارزمية تحديد التعبيرات المحذوفة ضمن المقطع B:

- لتكن U مجموعة تعابير البرنامج
- لتكن $\text{Supp}(B)$ مجموعة التعبيرات المحذوفة ضمن B ولتكن خالية في البداية
- من أجل كل تعبير $(x := y \text{ op } z)$ في المقطع B (بالترتيب)
- أضف إلى $\text{Supp}(B)$ جميع التعبيرات في U والتي تحتوي x بشرط ألا تكون محسوبة لاحقاً في المقطع

مثال:

لنفترض أن:

```
U={ 4*i, j+3, j+4, 2*j, t[k], a-1, t1-7, m+1, m-1, t[a], 2*a, l*3 }
```

وليكن لدينا المقطع B التالي:

```
t:=n-1;
i:=m+1;
j:=t[a];
k:=2*a;
t2 :=1*3 ;
t3 :=2*j ;
k:=t[k] ;
```


يؤدي تنفيذ الخوارزمية السابقة إلى ما يلي:

$Supp(B)=Empty$
 $Supp(B)=\{t1-7\}$
 $Supp(B)=\{t1-7, 4*i\}$
 $Supp(B)=\{t1-7, 4*I, j+3, j+4\}$
 $Supp(B)=\{t1-7, 4*I, j+3, j+4\}$
 $Supp(B)=\{t1-7, 4*I, j+3, j+4\}$
 $Supp(B)=\{t1-7, 4*I, j+3, j+4\}$
 $Supp(B)=\{t1-7, 4*I, j+3, j+4, t[k]\}$

- **تحديد التعابير القابلة للاستخدام:** نقول عن التعبير $(x \text{ op } y)$ أنه قابل للاستخدام في نقطة p من البرنامج في حال كانت كل المسارات من بداية البرنامج وحتى النقطة p تقوم بتقييمه واستخدامه، وفي حال لم تظهر بعد آخر استخدام له في أي مسار من المسارات أي عملية إسناد على قيمة x أو على قيمة y . لتنفيذ ذلك نستخدم الخوارزمية التالية:

خوارزمية تحديد التعابير القابلة للاستخدام:

١. لنسمي $In(B)$ مجموعة التعابير القابلة للاستخدام عند قبل الدخول في المقطع B ولنسمي $Ex(B)$ مجموعة التعابير القابلة للاستخدام بعد الخروج من المقطع B .
٢. لنحسب من أجل كل مقطع B المجموعتين $Prod(B)$ و $Supp(B)$.
٣. $In(B)=Empty; Ex(B)=Prod(B)$
٤. من أجل كل مقطع Bi مختلف عن B
 - $Ex(Bi)=U-Supp(Bi)$
٥. من أجل كل مقطع Bi مختلف عن B كرر حتى لا يعد هناك تغييرات
 - $In(Bi) = \bigcap_{P \in Predecessor(Bi)} Ex(P)$
 - $Ex(Bi) = Prod(Bi) \cup (In(Bi) - Supp(Bi))$

مثال: لتكن:

$$U=\{e1,e2,e3,e4,e5,e6,e7,e8\}$$

$(e1) \ t3:=m-1;$
 $(e2) \ t4:=m+n;$
 $(e3) \ k:=a+1;$
 $(e4) \ a :=u1 ;$
 $(e5) \ e :=2*c ;$
 $(e6) \ t1 :=t1+1 ;$
 $(e7) \ t2 :=t2-1 ;$
 $(e8) \ a :=u2 ;$

لنقم من أجل كل مقطع بحساب $Prod$ و $Supp$ لكل مقطع (البند ٢):

Bloc	Prod	Supp
B1	e1,e2,e4,e5	e6,e7,e3
B2	e5,e3	e6,e7
B3	e8,e2	e3,e5
B4	e1,e3	e6,e5

لنقم بحساب In و Ex في الحالة الابتدائية من أجل كل مقطع. وذلك بتطبيق البند (٣) ومن ثم البند (٤) على كل مقطع، فيكون لدينا:

Bloc	In	Ex
B1		e1,e2,e4,e5
B2		e1,e2,e3,e4,e5,e8
B3		e1,e2,e4,e6,e7,e8
B4		e1,e2,e3,e4,e7,e8

لنطبق الآن البند (٥):

Bloc	In	Ex
B1	Empty	e1,e2,e4,e5
B2	e1,e2,e4	e1,e2,e3,e4,e5
B3	e1,e2,e3,e4,e5	e1,e2,e4,e8
B4	e1,e2,e3,e4,e5	e1,e2,e3,e4

- خوارزمية حذف التعابير المشتركة العامة:

من أجل كل تعليمة قابلة للاستخدام $[(i) x:=y \text{ op } z]$ من المقطع B ولا يوجد أي تعديل على y أو على z قبل (i) في B:

- ابحث عن آخر حالات ظهور وحساب للتعبير $(y \text{ op } z)$ صعوداً في جميع مسارات شجرة البرنامج التي تؤدي للمقطع B.
- ولد متحول جديد u.
- بدل كل حالة ظهور وحساب للتعبير $[w:=y \text{ op } z]$ بالتعبيرين $[u:=y \text{ op } z; w:=u;]$.
- بدل التعليمة (i) بحيث تصبح $[(i) x:=u]$.